SoftwareX 13 (2021) 100625

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

A web-based simulation of discrete-event system of system with the mobile application DEVSimPy-mob

L. Capocchi *, J.F. Santucci

SPE UMR CNRS 6134 Lab., University of Corsica, 20250 Corte, France

ARTICLE INFO

Article history:
Received 28 February 2020
Received in revised form 31 August 2020
Accepted 6 November 2020

Keywords:
Discrete-event
Mobile application
Modeling
Simulation
Python
Web service

ABSTRACT

The paper proposes an original architecture to execute discrete-event simulations using mobile devices (in particular smartphones) and cloud. The originality of the proposed approach is highlighted by the way the connection between modeling and simulation environment, cloud and smartphones is performed via a solid web services oriented architecture. This approach has led to a software implementation based on the discrete-event system specification formalism which allows an explicit separation between the modeling aspect and the simulation features. Thanks to this explicit separation, the discrete-event models defined and validated by simulation are remotely accessible from smartphones via web services. With this new approach, a smartphone is part of the simulation model and it can be used for real data acquisition or to control the model during the simulation.

Code metadata

Current code version	v1.0
Permanent link to code/repository used of this code version	https://github.com/ElsevierSoftwareX/SOFTX_2020_63
Legal Code License	GPL v2.0
Code versioning system used	git
Software code languages, tools, and services used	python 2.x, Apache Cordova, JQuery-mobile, Ratchet, FusionCharts, Joint.js, Ionic, SoapUI, Ionic View, Mongo
Compilation requirements, operating environments & dependencies	
If available Link to developer documentation/manual	
Support email for questions	capocchi@univ-corse.fr

1. Introduction

Modeling and simulation (M&S) as a service (MSaaS) [1] is a concept based on the as-a-service model of cloud computing that combines web services and M&S applications. MSaaS frameworks offer effective simulation environments that can be deployed and executed on-demand by the modelers. The modelers discover new opportunities for working together and enhance their operational effectiveness, saving costs and efforts in the process of M&S. In a typical MSaaS platform, modeler can access to M&S functionalities as services by using browser or smart client. All the M&S services are stored in cloud and are accessible using smart clients that can embed web applications. The approach proposed in this paper is the same as any other typical MSaaS platform claims in terms of web services access. However, while the number of MSaaS tools is growing [1–6], they need to propose some important features in the field of dynamic simulations realization (add/remove models that change the model structure during the simulation) and the real data acquisition from sensors embedded to system of system (SoS) (like ubiquitous systems) involved in simulations.

On the other hand, M&S of SoS is a discipline dedicated to the field of engineering and research that tends to be exploited more and more by modelers, end-users and developers of mobile applications (or mobile apps) [3,7]. 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 (faults simulation in digital circuits, forest fires simulation, healthcare simulation, etc.). The analysis and development of a model and its simulator were therefore carried out by specialists whose working environment was confined to the workstation of a company or a research laboratory. The

* Corresponding author.
E-mail addresses: capocchi@univ-corse.fr (L. Capocchi), santucci@univ-corse.fr (J.F. Santucci).

https://doi.org/10.1016/j.softx.2020.100625

resulting computer programs were run locally using local input data (e.g. from a test bench). This unconnected mode of operation is no longer relevant with the advent of M&S involving new type of ubiquitous SoS that need to be accessible on-line [8]. Furthermore, this approach does not allow to provide simulation models that can be used to a large scale of non-specialist end-users.

The motivation behind this work is twofold: (i) to offer an effective environment for researchers in order to build and validate off-line discrete-event simulation models and automatically deploy these models on a on-line server which is remotely accessible from a mobile terminal (mobile application or responsive web application) for simulation (ii) to control and monitor simulation models from a mobile terminal which is used firstly to remotely interact with them (change model structure, start/stop/suspend simulation, modify parameters, etc.) via web services in a real physical dynamic environment and secondly to perform real data acquisition from both sensors embedded to SoS involved in simulation models or from the mobile terminal itself (considered as sensor) in order to collect real data like GPS position, image capture, etc.

This paper presents the DEVSimPy-mob mobile app that allows the web-based discrete-event simulation of ubiquitous SoS easily accessible by end-users. This mobile app improves the use of simulation models and makes them more available to a large scale of non-specialist users. The deployment plan of a model, from creation to use, is simplified by using the DEVSimPy/DEVSimPy-mob suite (the tool chain) as a generic collaborative framework to export simulation models that can be simulated from web. This tool chain cannot be categorized as a full MSaaS platform but it offers attractive MSaaS services related to real data acquisition and dynamic structure modification during simulation.

2. Problems and background

The proposed approach deals with a set of research questions involved by the connection between simulation, cloud and smartphones:

- How to interface simulation software with mobile application programming interfaces (APIs)?
- How to combine discrete-event simulation, cloud storage (with web services interfaces) and mobile terminals?
- How to use a generic approach to deal with these problems?

To address the three previous questions, a new web-based simulation approach including a collaborative discrete-event M&S environment associated with a mobile app in order to remotely simulate discrete-event models libraries via web services can be envisioned.

A possible approach can be defined from the domain of the design of SoS [9,10] based on web based discrete-event simulation. The generic proposed approach (Fig. 1) allows to automatically deploy a simulation model built by a team of engineers on a mobile terminal via web services (smartphone, tablet, etc.). When the system is modeled and validated (by simulation) in a M&S software, it can be proposed to the end-users to remotely simulate the resulting model of the studied system with data acquired from mobile terminal sensors immersed in a real physical environment. The benefit of the approach and its motivation are mainly highlighted through the ability (i) to perform data acquisition from mobile terminal sensors in order to simulate a system in a real physical environment (ii) to interact dynamically with the model from the mobile app during simulation.

The introduction of two previous capabilities needs a discrete-event framework based on an explicit separation between modeling and simulation parts. The DEVS formalism was introduced by Zeigler in the seventies [11] for modeling discrete-event systems in a hierarchical and modular way. 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 inputs over time and generates events as outputs over time is equivalent to a DEVS 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 models: (i) atomic models that represent the basic models providing specifications for the dynamics of a sub-system using state transition functions (ii) coupled models that describe how to couple several component models (which can be atomic or coupled models) together to form a new model.

An atomic DEVS model can be considered as an automaton with a set of states and transition functions allowing the state change when an event occur or not. When no events occurs, the state of the atomic model can be changed by an internal transition function noted δ_{int}. When an external event occurs, the atomic model can intercept it and change its state by applying an external transition function noted δ_{ext}. The life time of a state is determined by a time advance function called t_a. Each state change can produce output message via an output function called λ. A simulator is automatically associated with the DEVS formalism in order to exercise instructions of coupled models to actually generate its behavior.

DEVSimPy (Python Simulator for DEVS) [12,13] is a user-friendly interface for collaborative M&S of DEVS systems implemented in the Python language [14]. DEVSimPy has been set up to facilitate both the coupling and the re-usability of the DEVS models. Moreover, the DEVSimPy architecture is based on a MVC (Model-View-Controller) pattern coupled with the oriented aspect programming concept which renders the user interface and the simulation kernel independent. With DEVSimPy, DEVS models can be stored in a library in order to be reused and shared. A set of DEVS models constitutes a shared library due to the fact that all models can be loaded or updated from an external location such as from a file server which could be also considered as a kind of "on-line model store". The DEVSimPy M&S environment is used as a graphical environment dedicated to collaborative development and simulation of SoS using DEVS formalism [15].

We introduce DEVSimPy-mob [16] to give users the option of executing DEVSimPy models on-line from mobile terminals. The purpose of the DEVSimPy-mob mobile app is also to give users of the DEVSimPy environment the ability to simulate the models already defined onto DEVSimPy. The mobile then becomes a data source for the simulation models. Indeed, for example, considering a model that depends on some GPS system the user can select a model according to its position.

3. Software framework

3.1. Software architecture

According to Fig. 1, engineers use the DEVSimPy environment to implement DEVSimPy simulation models (in .yaml format) that they will store on the libraries file server (on the left part of Fig. 2)). These simulation models, which may depend on DEVS model libraries and plugins, can be simulated via the web using a command line version of DEVSimPy (DEVSimPy-nogui) software which is accessible through the REST (Representational State Transfer) [17,18] interface of a DEVSimPy-Rest server[1] (see

[1] https://github.com/capocchi/DEVSimPy_rest.

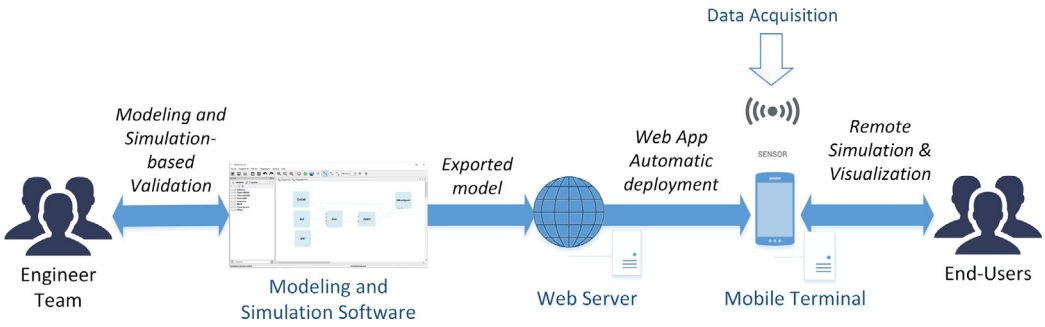

Fig. 1. Proposed generic approach allowing to interface and automatically combine M&S environment with cross-platform mobile app.

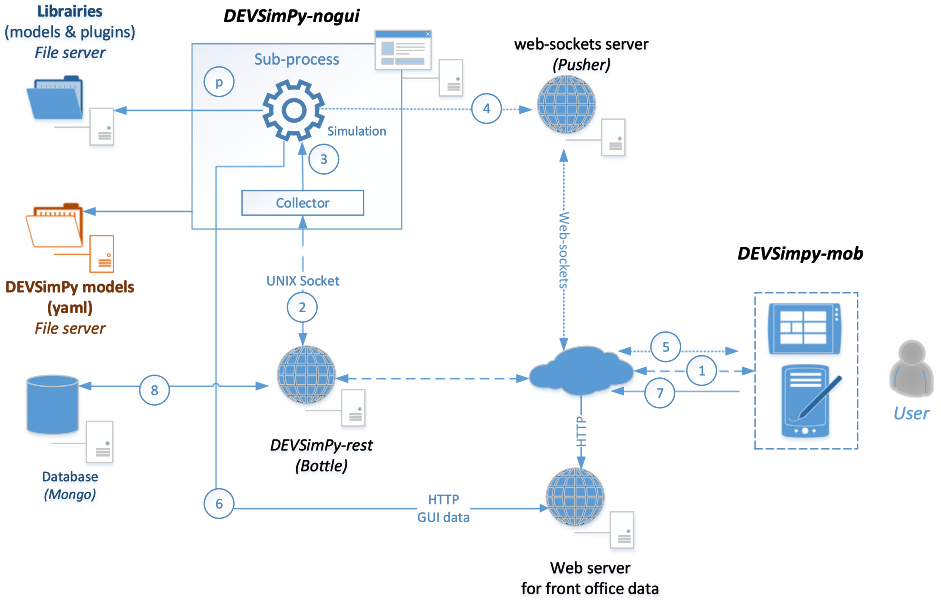

Fig. 2. Implementation of bi-directional communication between the DEVSimPy-Rest server and DEVSimPy-mob mobile app client: (p, 1, 2, 3) Simulation process initialization request using web services. (4, 5, 6) Simulation response to mobile app via Pusher services (6, 7). Simulation data results accessible for visualization. (8) Simulation data storing process in database (Mongo).

2 in Fig. 2). This interface is implemented in the mobile app DEVSimPy-mob and is accessible to all users (not necessarily experts of the M&S) wishing to exploit the simulation models.

From a technical point of view, the DEVSimPy-mob mobile app allows first of all to interact with web services provided by a REST web server. We note that DEVSimPy-mob does not have an off-line mode of operation. An Internet connection and a web server are essential for its use. The user of the DEVSimPy-mob app must therefore have such a web server which will be invoked with the URL when starting the application.

The interaction between the DEVSimPy-mob mobile app and the simulation process – modification of parameters in the "user to simulation" direction and visualization of results in the "simulation to user" direction – requires the establishment of a bi-directional communication between the mobile app and the REST web server (Fig. 2). In order to ensure the communication (see p, 1, 2, 3 in Fig. 2), the web service which receives a request for simulation creation, will create a process encapsulating the task corresponding to the simulation process and a task on receiving the messages (see p in Fig. 2). At the same time, it will create a UNIX socket to communicate with this receiving task. Thus, when the DEVSimPy-mob sends a request from an atomic model parameter interface for example (see 1 in Fig. 2), this request is transmitted by the web service to the reception task via the UNIX socket (see 2 in Fig. 2), and the receiving task in turn interacts with the simulation task via a shared memory embedded in a command line version of DEVSimPy (DEVSimPy-no-gui) (see 3 in Fig. 2).

Regarding the bi-directional communication, the Pusher [19] service has been used and allows to involve servers that do not support web-sockets. In fact, the simulation sends its data via HTTP requests to the Pusher server (see 4 in Fig. 2), which is responsible for re-transmitting them via web-sockets to subscribers (see 5 in Fig. 2). A first use of this communication is to send the progress of the simulation. Another use is to send the simulation results while simulation running. For this purpose, two new models have been created. The first directly transmits the results to DEVSimPy-mob via Pusher and the mobile app is then responsible for making these data accessible. The second uses a web service for the scientific representation of data: the Plotly [20] service. This collector sends to the Plotly server not only the data but also their representation mode (line, bars, 2D, 3D, etc.) and the server provides in return a URL where the data are accessible for visualization (see 6, 7 in Fig. 2). This URL, received at the collector level, is transmitted to the mobile app via Pusher. When a simulation is created, an identifier (ID) is associated and stored in a database (see 8 in Fig. 2). This ID is then used in the URLs that make it possible to carry out actions relating to this simulation.

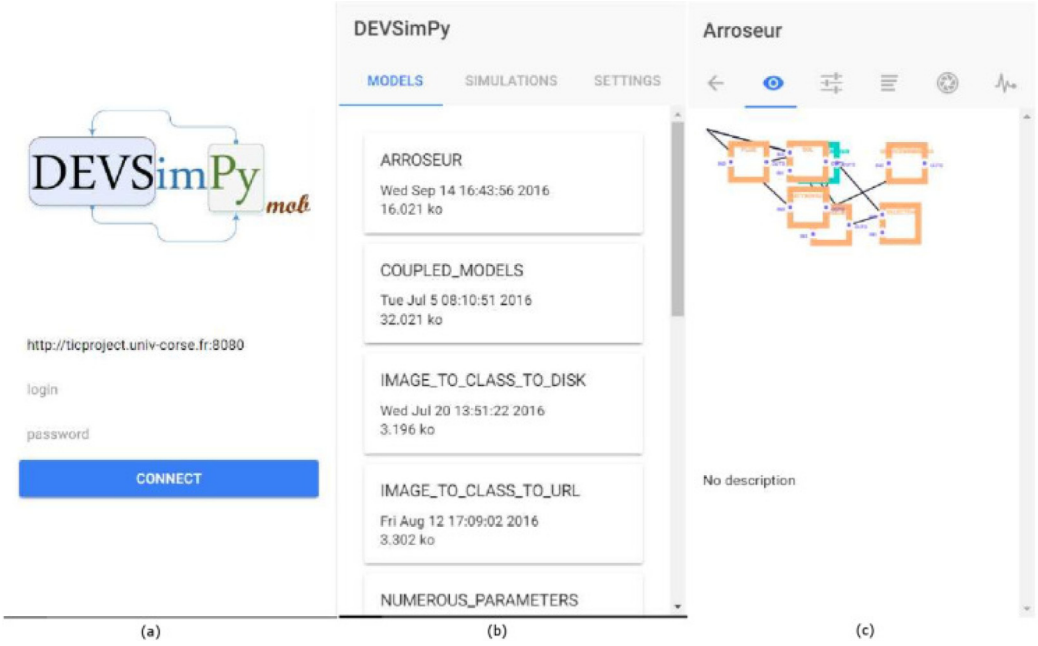

Fig. 3. Login and list of DEVSimPy models in the DEVSimPy-mob app: (a) The login interface (b) The list of DEVSimPy models (c) A diagram representing a selected DEVSimPy model "Arroseur".

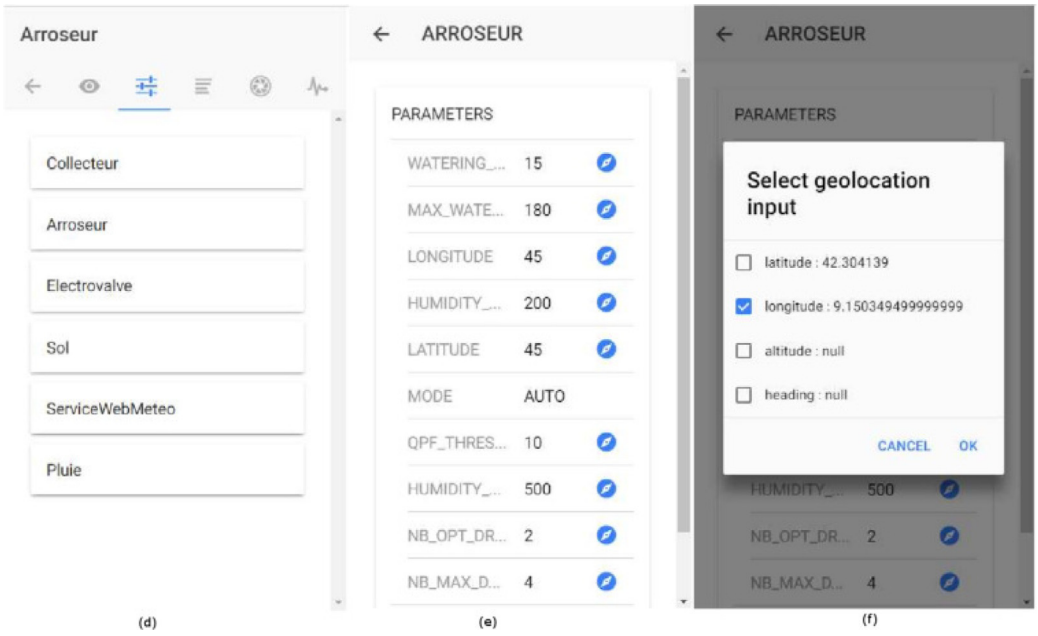

Fig. 4. Setting model parameters: (d) The atomic DEVS models included in the DEVSimPy model "Arroseur" (e) Setting of the atomic DEVS model "Arroseur" (f) GPS information from the sensor of the smartphone.

3.2. Software functionalities

The DEVSimPy-mob user interface is presented in Figs. 3–5. The URL of the REST web server is requested before the communication in the first login page (Fig. 3(a)). Then, a first level of tabs gives access to: the list of DEVSimPy simulation models (Models tab in Fig. 3(b)); the list of past and current simulations (Simulations tab in Fig. 3(b)); the server configuration (Settings tab in Fig. 3(b)).

A click on a specific model or on a past simulation (associated with a model) gives access to a second level of tabs:

1. The diagram of the yaml selected simulation model (Fig. 3(c)).
2. The list of atomic DEVS models included in the selected simulation model (Fig. 4(d)).
3. The list of past and current simulations of the selected model.
4. The state of the current simulation (not started, in progress, paused or finished). If the simulation is not started, we can choose the duration and the simulation can be started by clicking on the START button (Fig. 5(g)). When the simulation is in progress/pause, one will be able to suspend/resume the execution (Fig. 5(h)). When the simulation is finished, the execution report is displayed.

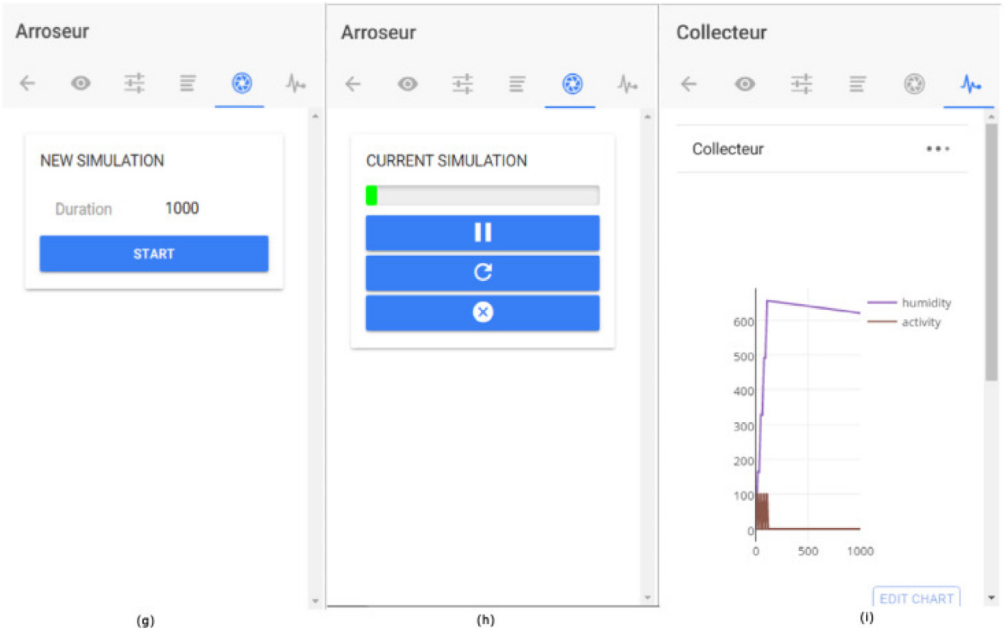

Fig. 5. Simulation settings and results for the model "Arroseur": (g) Starting interface of the simulation (h) Simulation progress (i) Simulation results stored in the model "Collector".

5. The visualization of the simulation results (Fig. 5(i)).

A mobile device can participate in a simulation through data acquisition using a click on an atomic DEVS model (from the list displayed as in Fig. 4(d)) which allows to reach a third level where are published the parameters of this atomic model (Fig. 4(e)). A click on the parameter makes it possible to define a new value or to access the sensor values of the mobile terminal (Fig. 4(f)) or to access the sensors (camera, GPS, etc.).

4. Implementation and empirical results

The DEVSimPy-mob mobile app is a hybrid application developed using the Apache Cordova© [21] technology that allows you to write a cross-platform application based on JavaScript, HTML5 and CSS [22] languages, and automatically generate the versions corresponding to the desired platforms (Android or IOS). In order to facilitate the evolution of the application, we chose to use Google's Angular framework which optimizes the structure and modularity of the code. We realized the development with the Ionic [23,24] platform which facilitates the integration of Angular with Cordova [25] that produces a browser version in addition to a mobile version. It provides basic graphical interface elements that are adaptable and compliant with the graphic charters of the different platforms.

The mobile app has been designed, modeled and implemented using the Ionic v2 framework and the architecture of the code is given in Fig. 6(a). The Ionic v2 framework integrates Angular and Cordova but it redefines also a new syntax and additional services: in particular the Page, Tabs and Tab components and the NavController service to facilitate navigation (managed as a stack of pages). Likewise, Ionic, via the ionic-native library, facilitates the integration of Cordova plugins. We can point out that Ionic offers a useful feature for sharing an application in development. The user-testers install the IonicView [24] app and access the latest version of the application uploaded by the developers.

The browser version of the DEVSimPy-mob (used in the development phase) can be obtained with the command 'ionic serve' (Fig. 6(b)). The www directory which contains the index.html file is completed by a www/build directory which contains the compiled code (all .ts files and 1 app.bundle.js file) and the html files. Then the command 'ionic run android' allows to create the version for Android that can be installed on the mobile terminal. Finally, the 'ionic upload' command can be used to upload the DEVSimPy-mob app to the Ionic platform and thus makes it available to users of the IonicView app.

The web services provided by the DEVSimPy-Rest server has been developed in Python language thanks to the Bottle [26] framework and respects the principles of REST architecture [27]. If, for example, the URL of the service DEVSimPy-Rest is http://ticproject.univ-corse.fr on port 8080, then:

- The GET request: http://ticproject.univ-corse.fr:8080/models returns a JSON stream [28] containing the list of available DEVSimPy models.
- The GET request: http://ticproject.univ-corse.fr:8080/models/model_id returns a JSON stream containing the description of the model identified by model_id.
- The POST request: http://ticproject.univ-corse.fr:8080/simulations loads the data in the format JSON ('model_name': my_model_id, 'simulated_duration': 100) makes it possible to create the relative simulation to the model my_model_id for a 100 DEVS simulation cycles.

Specifically, when creating a simulation, the web service creates a process from a command-line call to the DEVSimPy-nogui simulator with a selected model and a simulation duration. The web service records the data associated with a simulation (model name, date, duration, process identifier, etc.) in a Mongo [29] database. The unique identifier assigned to the simulation by the database server is embedded in the POST request. This identifier allows to access to the simulation simu_id by using the web service (http://ticproject.univ-corse.fr:8080/simulations/simu_id). All of the API resources are listed in Appendix and are available to external client including the DEVSimPy-mob mobile app. SoapUI [30] has been used to test them and the test suite is fully described by a single xml file available from https://github.com/CelineBateauKessler/DEVSimPy_rest/tree/master/test.

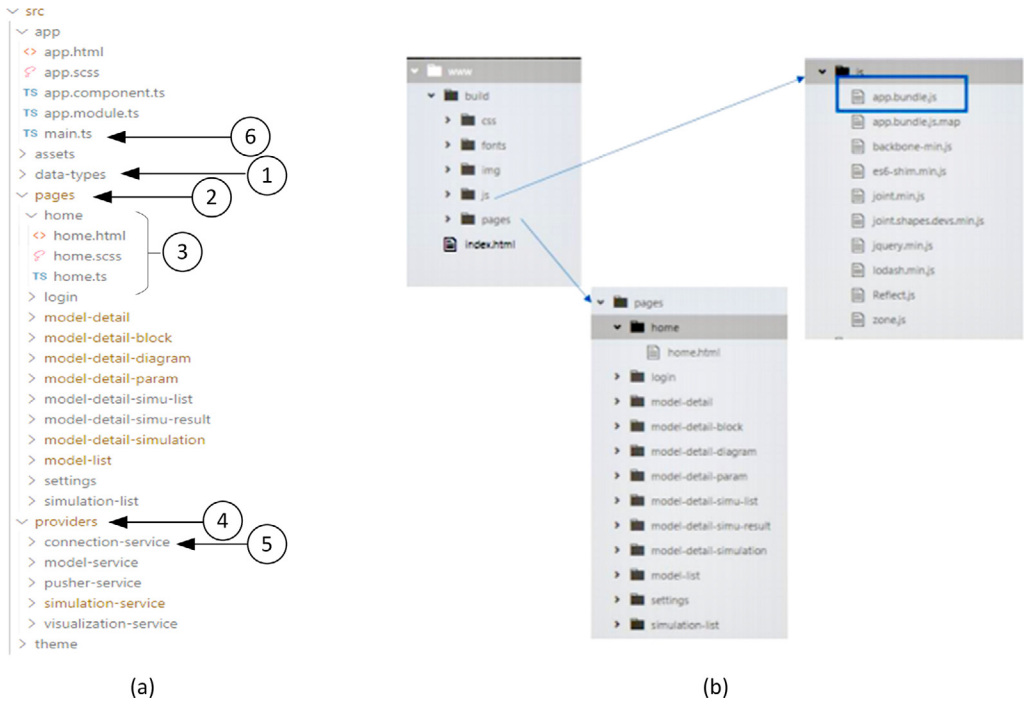

Fig. 6. (a) DEVSimPy-mob Ionic code architecture: (1) Model and simulation types definition (2) Ionic pages (3) Each page is composed of a template (.html), a style (.scss) and a controller (.ts) (4) Services (5) Each service has an associated .ts file (6) app.ts allows to define services that will be included in the pages and to define the first page of the mobile app. (b) Code structure after a build from Ionic serve command.

The advantage of using simulation tools via web services is not new [31–33]. In [33] 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 the traditional approaches to the modeling of SoS in the future. On the other hand, in [34] the authors emphasize the importance of simulation based on the use of web services but also the arrival of ubiquitous systems such as smartphones, tablets, etc. Of course this introduces issues involved in 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 allow to integrate the simulation as a service accessible by mobile terminal (smartphones) or to integrate mobile terminal (or components embedding sensors) as a source of data for the simulation [35].

5. Illustrative examples

The DEVSimPy/DEVSimPy-mob suite has been used to design, validate and interact with an ubiquitous system which is a smart sprinkler. This system is intended to optimize the amount of water consumed by watering using several input data such as soil humidity and weather forecasts. It uses a solenoid valve to watering. The user can view the evolution of the soil humidity rate, the activity of the solenoid valve and can force the start/stop of the watering using the DEVSimPy-mob mobile app.

The first step of the modeling process consists in describing the sprinkler component as an atomic DEVS model in the DEVSimPy environment. The state set is: active/inactive watering and on/off/auto mode. The input set is the soil humidity rate and the weather forecast. The output set is the on/off command for watering. In order to validate the decision algorithms, we must model the experimental frame of the sprinkler model with DEVSimPy by using a ground model (Ground in Fig. 7), a solenoid valve model (Electrovale in Fig. 7), a weather web service model (WeatherWebService in Fig. 7) and a rain model (Rain in Fig. 7).

The Ground model takes water flows at the input and produces soil humidity rate information considering a certain evaporation. The Electrovalve is a binary model: if it receives the command ‘on’, it produces a flow D; if it receives the command ‘off’, it produces a zero flow. Both Ground and Electrovalve models are combined in a coupled model called SensorActuator. The Rain and WeatherWebService models are generators that run scenarios described in a file (in date/message form). A test scenario consists in simulating a drought period with forecast rain (2 input files for Rain and WeatherWebService models).

Once the decision algorithms have been validated using simulation data, the next step is to integrate the real data and the interaction (command) with the simulation model via DEVSimPy-mob. To do this, each model of the experimental framework is replaced by its interface model — the Phidgets© SBC that embeds humidity and solenoid valve sensors (Single Board Computer) [36] for the SensorActuator model and a weather web service for the “WeatherWebService” which have been already implemented in libraries.

The Phidgets is an embedded system with a Linux OS of Debian type and a connection interface that can accept sensors. It is part of the family of components like ©,[2] Raspberry©[3] or Gadgeteer©[4] allowing to build communicating object systems. The Phidgets embeds a DHC (Dynamic Host Configuration Protocol) client, a web services server, an SSH server and can be accessible in “Remote” mode thanks to its ip address and a port which is by default port 5001. It is also possible to make it a wifi access point. There is an API in Python language which allows among other things to access the Phidgets in “Remote” mode and therefore to the sensors/actuators connected to it. It is this API which is used in the yaml models simulated by DEVSimPy-mob.

2 https://www.arduino.cc.
3 https://www.raspberrypi.org.
4 http://www.netmf.com/gadgeteer.

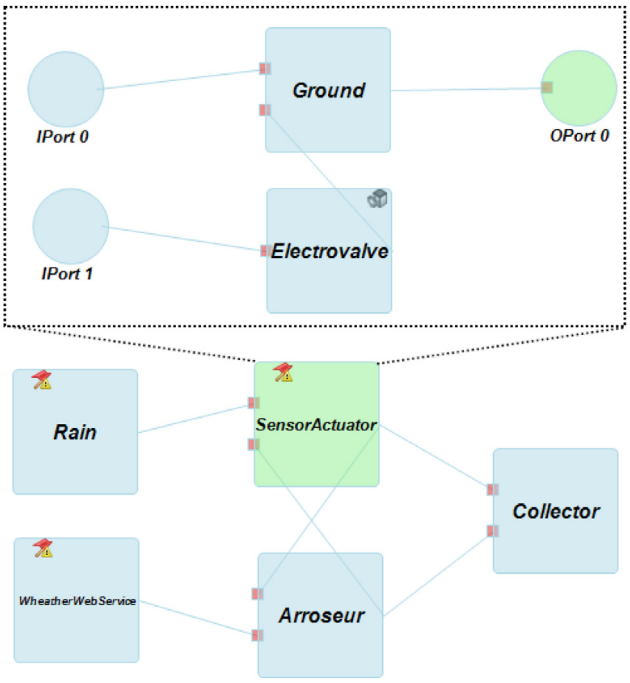

Fig. 7. DEVSimPy simulation model of the smart sprinkler system with the SensorActuator coupled model composed of the Ground and Electrovale (solenoid valve) atomic DEVS models.

In addition, we goes into real time simulation mode instead of simulated time and the simulation model works as an infinite simulation loop. The integration of the interaction with DEVSimPy-mob is automatic: once ported to the DEVSimPy-Rest server, the model is accessible to the user in bi-directional mode. DEVSimPy-mob allows you to view the activity of the sprinkler model using the collector (Fig. 5(i)) and allows the user to force watering by modifying the mode parameter of the "Arroseur" model during the simulation (Fig. 5(e)).

6. Discussion

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 apps through cloud storage and web services [3,7]. Recent developments in the cloud computing field and service-oriented architectures offer advances to better utilize M&S capabilities.

The proposed approach in this paper can be compared to the MATLAB © Web Apps one [37] that are web apps designed and compiled into the MATLAB software suite in order to propose browser-based web apps to end-users which are not MATLAB developers. MATLAB Compiler lets you share MATLAB Web apps with end-users who do not have MATLAB software. The Fig. 8 shows the workflow proposed by the MATLAB software suite dedicated to create a MATLAB web app. First off all, the MATLAB App Designer is used to implement the web app interface. Then, the code is compiled with a MATLAB compiler in order to deployed the complied package in a MATLAB Web App Server. The web app is available through an url and it can be shared via a classic browser. This approach needs to install previously the web app server.

The proposed approach differs from MATLAB by the possibility to inject real data to the simulation model settings using mobile terminal sensors. This is not possible with the MATLAB Web App solution which is browser-based. Concerning the development software development aspect, MATLAB needs to compile the Web App design in order to build an archive compatible with the MATLAB Web Server. With the proposed approach, there is no compilation phase, the model is automatically exported without transformation (compilation) into the Web server. The only thing that is required is to import all dependencies with the libraries of models. This task is achieved only once. Also when the Web App is developed, one need to have in mind some user features such as ability to change parameter values during running, etc. These difficulties in making changes are proposed by MATLAB but they do not depend on the real data coming from sensors as proposed by our approach. Moreover, displaying graphics in MATLAB App Designer requires a different workflow which can be included only in the MATLAB command line. With the proposed approach, the user can implement a large kind of observers (collector, QuickScope, etc.) in order to visualize its specific simulation results.

The DEVSimPy-mob proposed approach improves the use of simulation models and makes them more available to a large scale of non-specialist users. The deployment plan of a model, from creation to use, is simplified by using the DEVSimPy/DEVSimPy-mob suite (the tool chain) as a generic collaborative framework to export simulation models that can be simulated from web. This tool chain cannot be categorized as a full MSaaS platform but it offers attractive MSaaS services related to real data acquisition and dynamic structure modification during simulation. Accordingly, based on the literature consulted, no comprehensive tool exists that would allow a remote simulation of discrete-events models from mobile terminal with the addition of the possibility to inject real data in the simulation and to interact with the behavior and the structure of the simulation model always from mobile terminal during a simulation.

The mobile terminal becomes a source of input data for simulated models and allows the user to feed its simulations with real data. For example, initially the user can select a model depending on its position or the context in which it is located (mobility). So the selected model is dependent on real data that may be used by the simulation. Connected objects and mobile devices (smartphone) can communicate in a bi-directionally way with the simulation models. The possibilities offered by this new connectivity may be, for example, the validation of models from real simulation data. Conceptually, the proposed approach opens an interesting way around the methodology of M&S ubiquitous systems. Indeed, from the DEVS formalism point of view, the experimental framework is now real in the sense that the data of the simulation can be contextualized thanks to the use of the sensors embedded in the mobile devices or more generally in the connected objects.

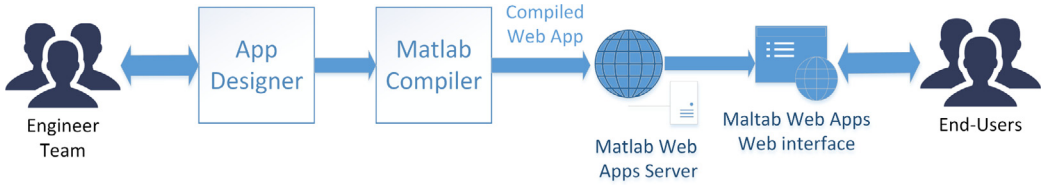

Fig. 8. Web apps with MATLAB Compiler workflow. App designers compile their design in order to deploy a MATLAB web apps stored in a dedicated MATLAB web server.

Table A.1
Request methods, URL ressource paths and expected results for the DEVSimPy-REST API.

CRUD action — Resource path	Results
GET - /models	Get the list of available models including their name, label, size and date creation
POST - /models	Model creation from attached file (YAML)
DELETE - /models/<model_name>	Model removing from its model_name
GET - /models/<model_name>	Get a JSON flow of the model_name model for representation and composition
GET - /models/<model_name>/atomics	Get the list of DEVS atomic model included in the model_name model
GET - /models/<model_name>/atomics/<atomic_name>/params	Get the JSON flow of all parameters as key/value for the atomics_name model included into the model_name model
PUT - /models/<model_name>/atomics/<atomic_name>/params	Update of parameters for the atomic_name DEVS atomic model of the model_name model using new values specified by {'param_name':param_value}
GET - /simulations	Get the list of simulations (running, suspended and finished)
POST - /simulations	Simulation process creation by given a new model_name and a new simulated_duration
GET - /simulation/<simulation_name>	Get JSON flow of the simulation status (RUNNING/PAUSED/FINISHED or UNKNOWN) from the simulation_name
PUT - /simulation/<simulation_name>/pause	Suspend the simulation specified by the simulation_name
PUT - /simulation/<simulation_name>/resume	Resume the simulation specified by the simulation_name
PUT /simulation/<simulation_name>/kill	Kill the simulation specified by the simulation_name
PUT - /simulation/<simulation_name>/atomics/<atomic_name>/params	Update of parameters for the atomic_name DEVS atomic model of the simulation_name simulation using new values specified by {'param_name':param_newvalue}
GET - /simulation/<simulation_name>/results/<result_filename>	Get the JSON flow for the visualization of simulation results as time/value
POST - /codeblocks	Model creation from code file (.py, .amd or .cmd)
POST - /blocks/<block_name>	Update of the code for the existing model block_name using the new specified input code file
DELETE /blocks/<block_name>	Code model removing from its block_name

7. Conclusion

This paper presents DEVSimPy-mob mobile app associated with the M&S DEVSimPy environment as a software suite for web-based discrete-event simulation from mobile app. Thanks to this approach, the DEVS models defined and validated by simulation using the DEVSimPy framework are remotely accessible by the DEVSimPy-mob mobile app. DEVSimPy-mob is based on bi-directional communication based on the use of web services that allow a user to control models before and during their simulation. This mobile app simulates DEVSimPy models – defined in a DEVS experimental framework – from real data that can come from platforms embedding sensors. Actually, the DEVSimPy-mob is available from GitHub [13]. It has been used to design, validate and interact with the ubiquitous system which is presented in the paper. Furthermore, the DEVSimPy-mob has also been used in the context of the problematic raised in the framework of Ballistic Missile Defence System in the United States [38]. We plan to evolve the DEVSimPy-Rest server by replacing the REST API with the new open source GraphQL [39] universal language in order to build a back-end and integrate it into the front-end more easily but also to make quick changes with less risk of crash. GraphQL is based on a simple idea: to move the assembly of a query from the server to the client. The interest is that DEVSimPy-mob sees the whole strongly typed graph of services instead of a multitude of REST services and builds the required request according to its needs.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors received no specific funding for this work.

Appendix. DEVSimPy-Rest API specification

See Table A.1.

References

[1] Procházka D, Hodický J. Modelling and simulation as a service and concept development and experimentation. In: Proc. of international conference on military technologies. 2017, p. 721–7. http://dx.doi.org/10.1109/MILTECHS.2017.7988851.

[2] Cayirci E. Modeling and simulation as a cloud service: A survey. In: Proc. of winter simulations conference. 2013, p. 389–400. http://dx.doi.org/10.1109/WSC.2013.6721436.

[3] St-Aubin B, Yammine E, Nayef M, Wainer G. Analytics and visualization of spatial models as a service. In: Proc. of the 2019 summer simulation conference. SummerSim'19, San Diego, CA, USA: Society for Computer Simulation International; 2019, p. 39:1–39:12.

[4] Zehe D, Knoll A, Cai W, Aydt H. Semsim cloud service: Large-scale urban systems simulation in the cloud. Simul Model Pract Theory 2015;58:157–71. http://dx.doi.org/10.1016/j.simpat.2015.05.005, Special issue on Cloud Simulation.
[5] Cayirci E, Karapinar H, Ozcakir L. Joint military space operations simulation as a service. In: Proc. of 2017 winter simulation conference. 2017, p. 4129–40. http://dx.doi.org/10.1109/WSC.2017.8248121.
[6] Bocciarelli P, D'Ambrogio A, Giglio A, Paglia E. Model transformation services for msaas platforms. In: Proc. of the model-driven approaches for simulation engineering symposium. Mod4Sim '18, San Diego, CA, USA: Society for Computer Simulation International; 2018, p. 12:1–12:12.
[7] Wang S, Wainer G. Modeling and simulation as a service architecture for deploying resources in the cloud. Int J Model Simul Sci Comput 2016;7(1). http://dx.doi.org/10.1142/S1793962316410026.
[8] Barbosa JLV. Ubiquitous computing: Applications and research opportunities. In: Proc. of IEEE international conference on computational intelligence and computing research. 2015, p. 1–8. http://dx.doi.org/10.1109/ICCIC.2015.7435625.
[9] Khaitan SK, McCalley JD. Design techniques and applications of cyberphysical systems: A survey. IEEE Syst J 2015;9(2):350–65. http://dx.doi.org/10.1109/JSYST.2014.2322503.
[10] Nielsen CB, Larsen PG, Fitzgerald J, Woodcock J, Peleska J. Systems of systems engineering: Basic concepts, model-based techniques, and research directions. ACM Comput Surv 2015;48(2):18:1–41. http://dx.doi.org/10.1145/2794381.
[11] Zeigler BP. Theory of modeling and simulation. Academic Press; 1976.
[12] Capocchi L, Santucci JF, Poggi B, Nicolai C. DEVSimpy: A collaborative python software for modeling and simulation of DEVS systems. In: Proc. of 20th IEEE international workshops on enabling technologies. 2011, p. 170–5. http://dx.doi.org/10.1109/WETICE.2011.31.
[13] Capocchi L. DEVSimPy, https://github.com/capocchi/DEVSimPy, Online; [Accessed 10 October 2019].
[14] Nagpal A, Gabrani G. Python for data analytics, scientific and technical applications. In: Proc. of amity international conference on artificial intelligence. 2019, p. 140–5. http://dx.doi.org/10.1109/AICAI.2019.8701341.
[15] Zeigler BP, Muzy A, Kofman E. Theory of modeling and simulation. third ed. Academic Press; 2019, http://dx.doi.org/10.1016/B978-0-12-813370-5.00003-1.
[16] Capocchi L. DEVSimPy-mob, https://github.com/capocchi/DEVSimPy_mob, Online; [Accessed 10 October 2019].
[17] Belkhir A, Abdellatif M, Tighilt R, Moha N, Guéhéneuc Y, Beaudry E. An observational study on the state of REST API uses in android mobile applications. In: Proc. of IEEE/ACM 6th international conference on mobile software engineering and systems. 2019, p. 66–75. http://dx.doi.org/10.1109/MOBILESoft.2019.00020.
[18] Segura S, Parejo JA, Troya J, Ruiz-Cortés A. Metamorphic testing of RESTful web APIs. In: Proc. of IEEE/ACM 40th international conference on software engineering. 2018, p. 882. http://dx.doi.org/10.1145/3180155.3182528.
[19] pusher group. Pusher, https://github.com/pusher, Online; [Accessed 10 October 2019].
[20] plotly group. Plotly, https://github.com/plotly/plotly.py, Online; [Accessed 10 October 2019].
[21] Camden RK. Apache cordova in action. first ed. Greenwich, CT, USA: Manning Publications Co.; 2015.
[22] Huang G, Liu X, Ma Y, Lu X, Zhang Y, Xiong Y. Programming situational mobile web applications with cloud-mobile convergence: An internetware-oriented approach. IEEE Trans Serv Comput 2019;12(1):6–19. http://dx.doi.org/10.1109/TSC.2016.2587260.
[23] ionic group. Ionic, https://github.com/ionic-team/ionic, Online; [Accessed 10 October 2019].
[24] Justin J, Jude J. Learn ionic 2: Develop multi-platform mobile apps. first ed. USA: Apress; 2017.
[25] Yang Y, Zhang Y, Xia P, Li B, Ren Z. Mobile terminal development plan of cross-platform mobile application service platform based on ionic and cordova. In: Proc. of international conference on industrial informatics - computing technology, intelligent technology, industrial information integration. 2017, p. 100–3. http://dx.doi.org/10.1109/ICIICII.2017.28.
[26] Framework BMW. Bottle. 2015, https://github.com/bottlepy/bottle.
[27] Sletten B. Resource-oriented architecture patterns for webs of data. In: Resource-oriented architecture patterns for webs of data. Morgan & Claypool; 2013.
[28] Martins JA, Mazayev A, Correia N. Hypermedia APIs for the web of things. IEEE Access 2017;5:20058–67. http://dx.doi.org/10.1109/ACCESS.2017.2755259.
[29] Chodorow K. MongoDB: The definitive guide. O'Reilly Media, Inc.; 2013.
[30] Nandan P. Mastering soapUI. Packt Publishing; 2016.
[31] Alfonseca M, de Lara J, Vangheluwe H. Web-based simulation of systems described by partial differential equations. In: Proc. of the winter simulation conference, vol. 1. 2001, p. 629–36. http://dx.doi.org/10.1109/WSC.2001.977348.
[32] Al-Zoubi K, Wainer G. Using REST web-services architecture for distributed simulation. In: Proc. of ACM/IEEE/SCS 23rd workshop on principles of advanced and distributed simulation. 2009, p. 114–21. http://dx.doi.org/10.1109/PADS.2009.16.
[33] Page EH, Buss A, Fishwick PA, Healy KJ, Nance RE, Paul RJ. Web-based simulation: Revolution or evolution?. ACM Trans Model Comput Simul 2000;10(1):3–17. http://dx.doi.org/10.1145/353735.353736.
[34] Taylor SJE, Khan A, Morse KL, Tolk A, Yilmaz L, Zander J. Grand challenges on the theory of modeling and simulation. In: Proc. of the symposium on theory of modeling & simulation - DEVS integrative M&S symposium. DEVS 13, San Diego, CA, USA: Society for Computer Simulation International; 2013, p. 34:1–8.
[35] Campillo-Sanchez P, Serrano E, Botía JA. Testing context-aware services based on smartphones by agent based social simulation. J Ambient Intell Smart Environ 2013;5(3):311–30.
[36] Greenberg S, Fitchett C. Phidgets: Easy development of physical interfaces through physical widgets. In: Proc. of the 14th annual ACM symposium on user interface software and technology. UIST '01, New York, NY, USA: ACM; 2001, p. 209–18. http://dx.doi.org/10.1145/502348.502388.
[37] Uran S, Jezernik K. MATLAB Web server and M-file application. In: 12th international power electronics and motion control conference. 2006, p. 2088–92. http://dx.doi.org/10.1109/EPEPEMC.2006.4778715.
[38] Kessler C, Capocchi L, Zeigler BP, Santucci J. Generic architecture for interactive mobile simulation of parallel DEVS models: A missile defense application. In: Proc. of winter simulation conference. 2017, p. 1515–26. http://dx.doi.org/10.1109/WSC.2017.8247893.
[39] graphql group. Graphql, https://github.com/graphql, Online; [Accessed 10 October 2019].