Simulation in the Cloud Using Handheld Devices

Emilio P. Mancini, Gabriel Wainer, Khaldoon Al-Zoubi, Olivier Dalle

Simulation in the Cloud Using Handheld Devices

Emilio Mancini[1] Gabriel Wainer[2]
[1]Laboratoire I3S UMR CNRS 7271
INRIA Sophia Antipolis – Mediterranée. France
{emilio.mancini, olivier.dalle}@inria.fr

Khaldoon Al-Zoubi[2] Olivier Dalle[1]
[2]Department of Systems and Computer Engineering
Carleton University, Ottawa, ON, Canada.
gwainer@sce.carleton.ca

Abstract— **In recent years, numerous applications have been deployed into mobile devices. However, until now, there have been no attempts to run simulations on handheld devices. We want investigate different architectures for running and managing simulations on handheld devices, and putting the simulation services in the Cloud. We propose a hybrid simulation and visualization approach, where a dedicated mobile application is running on the client side and the RISE simulation server is hosted in the Cloud. In particular, with our prototype, we explore the remote management of a simulation tool using a dedicated native application running on an Android Smartphone, and showing the evolution of a simulation model for a forest fire spread, mashing-up the generated graphics with online GIS services.**

Keywords; simulation, management, RISE, handheld devices

I. INTRODUCTION

On-line simulations can be useful for the live analysis of phenomena field. Examples of such phenomena include swarm movements in rural areas, in which a field-expert needs to update the simulation models at run-time with current observed data (such as the wind) [1], or civil protection scenarios, like forest fire spreading, in which fire troopers can use simulation models to predict fire propagation in real-time, based on their field observations and weather forecast information [2]. Indeed, due to their complexity, most simulations require high performance computing infrastructures. For this purpose, computational cloud may prove to be a scalable and cost effective back-end solution, especially when using a parallel or distributed simulator [3].

Web-based simulation has become a popular technique to conduct online simulations. The idea is to use a Web server and to implement the networked architecture using a particular architecture (client-server, High Level Architecture, CORBA, RPC, etc) [4]. Many advanced implementations have been built on SOAP Web Services to communicate [5] [6]. These simulation middleware are complex to interoperate, and their composition scalability is limited. Instead, RESTful Web Services can solve these issues by imitating the Web interoperability style. Instead, the Representational State Transfer style (REST) focuses on the resources more than on the operations, solving the interoperability limitation and making easy the development of Mash-Ups [7] (which reuse and combine existing services to build a new web application). Based on these ideas, in [8] we introduced the first existing RESTful Interoperability Simulation Environment (RISE) middleware.

In recent years, numerous applications have been deployed into mobile devices, as they have become popular tools, assisting their users by performing a myriad of personal tasks. They are portable, available everywhere, may be equipped with advanced hardware features (a camera, a GPS, a compass, gyros and accelerometers). Nevertheless, until now there has not been any attempts include mechanisms to run simulations on these devices. Despite the enormous improvement in Smartphone technology, computational capability in mobile devices is still not powerful enough to allow them to run complex simulation models. We want to investigate different architectures, putting simulation services in the cloud and using the simulation results into the Smartphone. This solution also has the advantage of assuring that the same simulation data can be retrieved from different devices, independently of hardware or even geographic location.

As discussed in [9], the distribution of computations between the simulation results and the visualization can run in a remote server (remote simulation and visualization), both of them can be on the client (local simulation and visualization), or a hybrid approach can be used (i.e., the simulation is hosted on a remote server and the visualization is generated locally by the client). However, implementing such web-based simulations on a handheld device (using the results of computations coming from one or several cloud computing facilities, and mashing them up), usually requires time and expertise.

In this article we introduce the first existing Smartphone simulation client using a RISE server hosted in the Cloud to run the simulation experiments. We are interested in reducing both the design and implementation time, and allowing a better integration with the devices. We propose a hybrid client/server approach, where a dedicated mobile application is running on client side and the RISE on the Cloud. This flexible approach is particularly suitable for handheld devices because of their limited resources (memory, energy, and computation) and network constraints. Our proposed approach requires the development of a dedicated front-end application in order to use the specific features of the device, like the GPS or other sensors. This dedicated front-end application needs also to be integrated in the RISE middleware using the standard API available on the device. On the basis of these two points, the application can setup the simulation using the information taken from the device's sensors.

As demonstrated by the implemented prototype, the proposed approach actually optimizes the design and the implementation of systems for the remote access to a simulator using handheld devices. In particular, with our prototype, we explore the remote management of a simulation tool using a dedicated native application running on an Android Smartphone, and showing the evolution of a simulation model for a forest fire spread, mashing-up the generated graphics with online GIS services. We apply a hybrid simulation and visualization pattern, using the onboard GPS to setup the model, then caching and post-elaborating the raw results.

II. BACKGROUND

As discussed in the introduction, the integrated simulation system presented in this paper uses the RESTful Interoperability Simulation Environment as middleware to interface the simulators [8] [10]. The REST style is, with SOAP, one of the leading techniques used in services architectures. REST focuses on the resources more than on the operations assigning an URI, a state and a representation to every resource. Webber, Parastatidis and Robinson, in [11], give a detailed description of it, while Riva and Laitkorpi, in [12], investigate how to design REST based services dedicated to mobile devices. The objective of RISE is to allow a plug-and-play interoperability paradigm for simulation tools. The way it achieve this goal, is decoupling the services from the required formalism. In other words, it provides a general purpose API to interface different simulators. In this way, the clients can use an experimental framework with a number of instances with different settings. This approach allows two or more instances to exchange messages with each other. One of the driving ideas of this system is to make every experiment persistent and repeatable, by storing simulation configurations. RISE exposes the tools through a set of URIs, so they are dynamically accessible from anywhere. When a client makes a request, it uses a specific URI. RISE looks for the resource, and sends the request to it, collecting and forwarding the response to the client. Clients connect to the URIs via HTTP channels. They use GET operations to read resource data, PUT operations to create new resources or update them, and DELETE to remove a resource. For example, a simple session could include the following sequence:

1. `PUT http://.../cdpp/workspaces/bob/DCDpp/model`, to create a model in the workspace "*bob*";
2. `POST http://.../cdpp/workspaces/bob/DCDpp/model?zdir=files`, to submit the model files;
3. `PUT http://.../cdpp/workspaces/bob/DCDpp/model/simulation`, to start the simulation;
4. `GET http://.../cdpp/workspaces/bob/DCDpp/model/results`, to download the results

A complete description of RISE can be found in [10] and [8]. Although RISE can manage different simulators, we have a using CD++, a simulation toolkit based on the DEVS and Cell-DEVS formalisms. More information about it can be found in [13]. The most common way to use a simulator remotely, is by designing a web-based tool. Byrne, Heaveya and Byrne in [9] provide an extensive review of web-based simulators, discussing the advantages and disadvantages of the client/server pattern applied to the simulation, and analyzing how and with which technologies it can be implemented. In the light of their paper, our work uses a hybrid simulation and visualization approach to drive a remote managed simulation (the client grabs the raw data and visualizes them). Once a simulator is exposed as service, it can be composed with other complementary services. For example, Harzallah, Michel, Liu and Wainer, in [7], propose a composition of heterogeneous simulation and presentation services for emergency planning teams. They integrated a set of web-services for a forest fire spread simulation, for a global weather service, and a Geographic Information Systems. However, this requires dedicated implementation for the integration interface, and a considerable design and implementation effort (approximately 2160 person/hours).

Cloud patterns can also be helpful when providing such services, ensuring scalable and cost effective solutions. Fujimoto, Malik and Park, in [3], explore how to exploit the potential of the cloud computing on the simulation purposes. However, they point out that parallel simulations often require an intense exchange of small messages, comporting performance degradation in the dynamic cloud environment. Also D'Angelo, in [14], draws attention to the performance drawbacks of the cloud environments when using the simulation as a service, proposing an approach based on multi-agent systems.

Web simulation systems present their results online. For example, Boukerche *et al.* in [6] propose to show the results with a visualization framework, using the X3D standard. They directly interface the visualization module with an HLA federation [15]. However, although it is possible to access those interfaces through every web browser, the use of dedicated applications may improve the quality and the easiness of the simulation management. In particular, when the visualization and user's interaction are moved on the client terminal, or when the device can collect the simulation parameters without the need of a separate sensor network. For example, Holmes and Kawalsky, in [16], explore the management of remote simulations using handheld devices. They present the interface of an eScience simulation engine with a PDA/Smartphone client. They split the visualization modules between the device and the server: in other words the Smartphone does not receive the actual simulation results, but visualization-oriented information. We want instead to decouple the client application hosted on the device. Huynh, Knezevic, Peterson and Patera present a related research in [17]. They interface a large-scale parallel application with thin devices. They analyze the decomposition of the simulation model in an offline component, to be run on a supercomputer, and in an on-line part to be deployed on a Smartphone. Instead, we propose a general architecture to efficiently integrate and manage remote simulators with handheld devices. Similar solutions have been already studied. For example, Holmes and Kawalsky in [16] propose a handheld interface for a simulation engine. However, in their approach, the clients need the support of a server module sending pre-elaborated visualization data. Our solution, instead, tries to decouple the client from the server to better handle less efficient networking infrastructures. Huynh, D. Knezevic, J. Peterson and A. Patera have done another related study in [17]. They propose a system to interface a handheld device to a supercomputer, exploring how to split a PDA problem into an offline and an online part. However, their solution is specifically targeted for scientific large-scale real-time simulations. We aim, instead, to provide a general architectural solution to exploit simulations on handheld devices efficiently.

III. MANAGEMENT OF SIMULATION SERVICES WITH HANDHELD DEVICES

Smartphones are now pervasive, and producers continuously improve their interfaces to make them easier to use and less error-prone. This technology now allows us to execute com-

plex simulations remotely. This approach offers many benefits to the users, among others the most obvious is usability and ubiquitous availability. The Smartphones and handheld tablets have a very small size, which allows using the simulators in situations where other systems (desktops and laptops), could be impractical. Another advantage is the cooperation opportunity, allowing a group of nomadic operators to concurrently setup a simulation model running on a server. However, the cooperation and the ubiquity may lead to scalability issues or to difficulty for server dimensioning. So the use of an Infrastructure as a Service (IaaS) and the service interface of RISE, can greatly advantage these systems.

Despite current Smartphones can have high frequency dual core CPUs and relatively generous memory amounts, it is more convenient to execute time-consuming simulations on more powerful hardware and remotely control them. The simulation management using handheld devices raises interesting challenges due to their limited computational, memory and connection resources. Therefore, the use of client/server patterns with the Smartphones needs to be well analyzed in order to guarantee the required interoperability and usability, mainly due to the connection instability, to the energy consumption, and obviously to their size. A purely web-based client/server system requires a persistent connection to interact with the services, and to visualize the data, but nomadic users may experience often the loss of the connection, especially when they are interested to use the simulation analysis from rural areas. Sometimes the connections are not stable, and often their latency is much worse than the latency of other wireless networks. Therefore, to design an architecture with the purposes discussed in the introduction, we need to choose patterns and technologies that can handle these specific characteristics. In particular, the study presented here integrates two existing tools: CD++ and RISE. Integrating a client with this software stack requires multiple stages.

shows a sequence diagram describing them:

a) The user sets up the parameters to run the simulation;
b) The application takes some of the parameters directly from the device's sensors. In this study, we use the GPS data, but, on the basis of the simulation model, other sensors can provide useful information, as cameras, microphone, accelerometers or timers;
c) The users starts the simulation;
d) An active data connection is needed to send an http message to the RISE server. The use of the http protocol is of great advantage, as every modern mobile operating system development kit supports it, and it is usually not filtered by network's security policies. The simulations are executed off-line, taking advantage of the REST protocol, that allows the state persistence between clients' requests [11];
e) The simulation results are sent to the client, after it requests them;
f) Storing these results on local file systems, is better than download on demand, because the high latency, the limited bandwidth and the instability of mobile connections;
g) How to present effectively this information to the user is strictly dependent from the simulation model. For example, in some case one can take advantage of the maps support of most modern mobile platforms, drawing the simulated data on them. In this schema, the Smartphone asks a GIS server [18] for a map with the same coordinates and region of the simulating area. It can then overlap the map with symbols and colors in order to enrich it with the computed data.

One interesting scenario is the execution of location-aware simulations. Most modern Smartphones have an integrated GPS receiver, which can be used to identify the event's location, and send it to the simulation server in order to properly chose or setup the model. Another interesting point of the proposed schema is the energy savings it allows to achieve, minimizing the wireless communication, due to the use of the asynchronous REST calls.

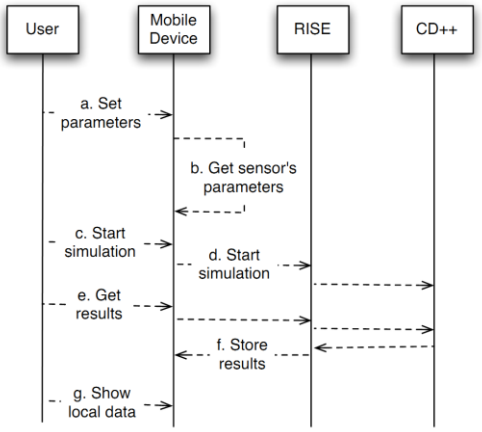


Figure 1. Sequence diagram for the simulation management on a Smartphone.

A. The Architecture

The platform presented here, integrates several different subsystems, each one implemented with different technologies. Our high-level architecture hosts the management and presentation modules on the mobile devices, and the simulation modules on the back-end infrastructure.

At the lowest level of our architecture, a simulator executes the simulation models. The RISE middleware drives the simulator and collects the results, sending them to the user's Smartphones. In this architecture, the client side has three main duties: set-up and start the simulation, download the results for off-line use, and present them to the user.

A different module manages every one of these points (see Figure 2), and all them are glued by a model-view pattern with the actual user interface. The first step requires the collect of user's data and their codification in an http message using the syntax requested by the server. Some of the simulations that could be interesting to run on Smartphones are related to the user's positions. In other words, a user may be interested to know the results of a simulation concerning the site from where he sends the request, for example, to forecast the local evolution of a forest fire or a flood. In these cases, the simulator should run the model with the

users' geographic coordinates as starting point. This information is provided in most Smartphones by an internal GPS, and it is automatically handled by the client application.

Once the simulation data are ready, the user can download them. This operation, however, is expensive in terms of resources for a mobile. The download operation may block the user interface during the transfer operations, for this reason, all the REST network operations require a multithread approach. However, the mobile operating systems have some additional constraints. For example, every user interface interaction must be done in a specific thread (the main one). For this reason, the network threads require an advanced model for the synchronization with the other application's modules. In particular, after selecting the model, the application can start the simulation, executing a concurrent task to send the appropriate REST message to the server. It calls in this way the RISE API to check if the simulation is still running, and when it ends, the user can download the raw data. The choice to download the data for an off-line usage is due to the need to optimize the response time of the presentation module. This one should be able to present the data in ways that are easy to understand and to interact.

Once the application stores the raw data locally, it needs to transform them from the simulator output syntax to a format appropriate to the small display size. For this reason, in this step it creates a set of bitmap images, one for each simulated time increment. For the experiment with a forest fire model, every one of these images is filled as a grid matching the simulation model's grid. This means that if the simulation model represents an area with a grid of *30x30* cells, the visualization module will group the result image's pixels in *30x30* cells, and will color every cell with a palette whose shade gradations are proportional to the simulated values. For example, we modeled the maximum forecasted fire value as semitransparent red, the minimum as semitransparent green, and no fire with completely transparent pixels (Figure 3b). Once the frames are ready, the application asks a GIS server for a map matching the same coordinates. The user interface modules, at this point, overlays the map with the computed bitmaps, following the user requests. For example, in our prototype, the user can go through a series of consecutive synthesized images to analyze the evolution of a forest fire.

For the prototype purposes, we used two separate databases, the first one to feed the simulation model and the second one to display the maps (using a public map server). This model, however, has the downside that the geographical data for the simulation and presentation modules are not synchronized. Although this design improves the communication and the resources requirements for the server, and the design and implementation time, the two databases does not precisely match and the user has no control over the public GIS data neither they knows how updated they are. For this reason, we plan to extend the architecture including GRASS as a dedicated GIS engine [19], using one of the several studies proposing a Grid or a Web interface for it, for example the one proposed by Brauner in [20].

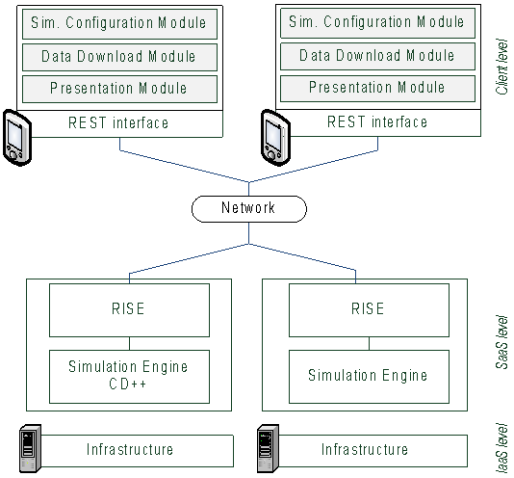


Figure 2. Architecture: a client/server pattern and RESTful interface to provide interoperability between services and mobile devices.

B. A Case Study on the Android Platform

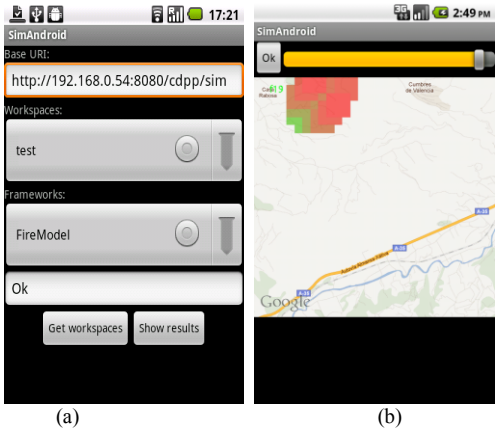
We developed the prototype using the Android platform [21]. The applications built with this development kit, use the concept of Activity as one of their interface basic blocks. From the user perspective, an Activity can be simplified as a single screen form in the application. Therefore, our mobile application is designed as a set of Activities matching the proposed architecture. In other words, the implementation associates a specific Activity to every component of Figure 2: the *configuration*, the *download* and the *presentation* modules. The *Simulation Configuration Module* gathers data from the device's sensors (and from the user) in order to choose and setup the simulation model. The current implementation takes the location from the GPS sensor. It can use different providers for the location data, taking it from the memory (last known position), from the network infrastructure or from the satellites. As the most precise provider is also the slowest to provide the position, the current implementation uses a combination of the three, refining the coordinates as the data become available. The Configuration Module also gathers other required information from the user, including the simulation workspace. Once all data are available, it can setup and start a remote simulation using the RISE service. To communicate with the server, the client includes a set of reusable Java classes. Their goal is to open a background http connection and then execute the PUT, POST or GET operations. These classes are implementations of the `android.os.AsyncTask` abstract class [21], which allows to perform background operations and to call methods interacting with the user interface (the Android's GUI is not updatable in other than the main thread). In this way, long network operations can avoid making the device unresponsive, and at the same time, the application can provide some feedback to the user. An example of a client code snippet using the RISE service API is the following:

```
private class GetWorkspaces extends AsyncTask {
 protected String[] doInBackground(String url) {
    HttpClient client = new DefaultHttpClient();
    HttpGet req = new HttpGet();
    req.setURI(new URI(wsUrl));
    HttpResponse response = client.execute(req);
    ...  } ... }
```

In this example, we can see that we first create an http client object (`HttpClient` class) and we issue a GET request (`HttpGet` class). This request is then set to be sent to the specific URI (`setURI` method), and calls the RISE service, which launches the simulation and waits for the operation results (`execute` method). It should be noted that the software development kit provides all the base communication classes, minimizing in this way the development effort.

The *Data Download Module* executes a GET operation over an http channel to retrieve the simulation results and store them in the local file system. In order to optimize communication, RISE compresses the information, so the Data Download Module needs also to decompress the data as soon as it receives them. The screenshot in Figure 3a shows the user interface associated to the *Data Download Module*. It lets the users to chose a simulation server, a workspace (with the aim, for example, to group several models), a simulation model, and then to download the result data. The graphical widgets here match the RISE URIs, allowing the users to visually access the right simulation. In other words, every widget of this form concurs in building the right URI for the RISE server. An instance of `AsyncTask` takes this URI, builds an http GET request, sends it to the RISE service, and waits for the results. When they arrive, it decompresses them and updates a progress bar on the device screen, to give some visual cues to the user. The *Presentation Module* interprets the information received from RISE, synthesizes a graphical representation, and shows it over a map. This module translates the data creating off-line a sequence of images, which the user can surf to see graphically the phenomenon evolution. In the example shown in Figure 3b), the simulator outputs a file describing the variations over the time, $\Delta_t(r,c)$, of a matrix $v_t(r,c)$ representing the amount of the fire in an area (here, r is the row and c the column of the matrix). Starting from a first matrix, the client computes the next one adding the value from the raw simulator output: $v_t(r,c)=v_{t-1}(r,c)+\Delta_t(r,c)$ Then, it creates an image from every matrix, whose pixel are computed as $p(x,y)=f(v(r,c))$ with $c=x \bmod |w/n_c|$, and $r=y \bmod |h/n_r|$. Where w is the image width, h is the image height, n_c and n_r are the number of rows and columns of the matrices. $f(.)$ is the mapping function from the matrix value to the pixel color. It takes a single value, and returns a vector of four values that are the color components red, green, blue and alpha (transparency). Figure 3b shows the last frame of a simulation of a forest fire (the area in the top-left) [22].

It is important to note that the availability of RISE makes the implementation much simpler, errors are easier to find (due to the well defined methods in the interface), and the development of a mashup application like the client presented here can be done in a more effective way. Compared to the mashup presented in [7], which required a dedicated implementation for the integration interface and a considerable implementation effort (approximately 2160 person/hours), the current clients (which can easily adapted to other applications) can be developed in a shorter period of time (this particular application took only 375 person/hours).



(a) (b)

Figure 3. Interface on Android devices: fire simulation model for a given geographical area.

As discussed earlier, the transmission rate of wireless networks can introduce problems in this kind of distributed application. As we know, performance of these networks can vary noticeably, and a terminal could change the network type on the fly. However, the analysis on GPRS, EDGE and UMTS network is indicative of the average performance on the most common situations. In order to analyze these issues, we tested the system using a simulation service hosted at Carleton University, and a mobile client located in France. Considering only the network effects, the theoretical minimum required time in this case is approximately 120 s on GPRS, 40 s on EDGE, and 5 s on UMTS. However, the server, the back-end network and the device introduce latency and reduce the bit rate.

Figure 4 summarizes the obtained results: the histogram compares the download times of a 1.2 MB simulation result file over GPRS, EDGE and UMTS networks, using a device with 128 MB of RAM and a 600 MHz CPU. The time in the UMTS network is only slightly lower than the time over EDGE. This effect is related to the CPU speed of the device. The server is designed to use efficiently the network, so it compresses the whole results and makes it available through the REST interface. The clients have to decompress it while downloading, so this factor automatically limits the network usage. Obviously, this is strictly related to the particular used device. The figure also compares the experimental data with the theoretical download time using the specification's full bandwidth reported in Table 1. It is interesting to note that the data encoding may compromise the efficiency of the data transfer, when using low-end devices. In other words the CPU time required to decode the data and store them on the local file system, could affect the downloading time on slower devices.

Standard	Bitrate(Mb/s)	Latency(ms)
GPRS	0.040/0.080	150-550
EDGE/EGPRS	0.1184/0.2368	80-400
UMTS/3G	0.128/1.920	35-200
IEEE 802.11g (WiFi)	54	50
IEEE 802.16e (WiMAX)	96	25-40

Table 1. Specification of bit rate and latency for wireless network for mobile devices. The actual values may vary also during the same session.

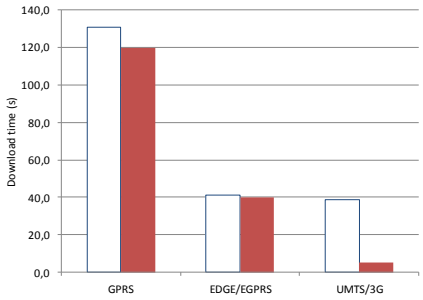


Figure 4. Experimental (left) and theoretical (right) minimum download time on different networks.

IV. Conclusion and Future Work

Web based simulations are widely used; however in some situations they have some disadvantage, for example when a good quality connection is not available as in the case of their management using handheld devices. In this paper, we propose an architectural solution to integrate a heterogeneous simulation system minimizing the design and development effort. This solution employs handheld devices and the RISE middleware to drive computational heavy remote simulations. We used a hybrid simulation and visualization approach, in which the native client application sets the initial model parameters and builds the output in a user-friendly way, while the simulation service make available the raw results using a REST interface. As result of this study, we presented a prototype analyzing the results of a forest fire model running on a RISE server. Our test application identify the user's location using his onboard GPS, runs the appropriate model, downloads the results and synthesize a sequence of images to display the forecasted model evolution over a GIS map. A future version of the prototype will include the logic for computational steering [23]. Moreover, exposing a simulator as a service (Software as a Service layer in a cloud infrastructure, SaaS), is important from an integration point of view. However to allow more scalability or advanced scenarios as the pay-per-use, it needs also a dynamic infrastructure (Infrastructure as a Service, IaaS). A future study will delve into this aspect.

Acknowledgment

This work has been co-funded by the DISSIMINET Associated Team initiative (INRIA Sophia Antipolis MASCOTTE project-team and ARSLab; Carleton University), and NSERC (Canada).

References

[1] C. M. Topaz, *et al.* "A model for rolling swarm of locusts," *European Physical J. Spec. T.,* no. 157, pp. 93-109, 2008.

[2] L. Ntaimo, X. Hu and Y. Sun, "DEVS-FIRE: Towards an Integrated Simulation Environment for Surface Wildfire Spread and Containment," *Simulation,* vol. 84, no. 4, pp. 137-155, April 2008.

[3] R. M. Fujimoto, A. W. Malik and A. J. Park, "Parallel and Distributed Simulation in the Cloud," *SCS Modeling and Simulation Magazine,* vol. 1, no. 3, July 2010.

[4] G. Wainer and K. Al-Zoubi, "An Introduction to distributed simulation," in *Mod. and Sim. Fundamentals,* Wiley, 2010.

[5] S. Mittal, J. L. Risco-Martín and B. P. Zeigler, "DEVS/SOA: A Cross-Platform Framework for Net-centric Modeling and Simulation in DEVS Unified Process," *Simulation,* vol. 85, no. 7, pp. 419-450, July 2009.

[6] A. Boukerche, F. M. Iwasaki, R. B. Araujo and E. B. Pizzolato, "Web-Based Distributed Simulations Visualization and Control with HLA and Web Services," in *Proc. of the 2008 12th IEEE/ACM DS-RT*, Washington, DC, USA, 2008.

[7] Y. Harzallah, V. Michel, Q. Liu and G. Wainer, "Distributed Simulation and Web Map Mash-Up for Forest Fire Spread," in *Proc. of the 2008 IEEE Congress on Services*, USA, 2008.

[8] K. Al-Zoubi and G. Wainer, "Distributed Simulation Using RESTful Interoperability Simulation Environment (RISE) Middleware," in *Intelligence-Based Systems Engineering*, Springer Berlin Heidelberg, 2011, pp. 129-157.

[9] J. Byrne, C. Heaveya and P. Byrne, "A review of Web-based simulation and supporting tools," *Simulation Modelling Practice and Theory,* vol. 18, no. 3, pp. 253-276, 2010.

[10] K. Al-Zoubi and G. Wainer, "Rise: Rest-ing heterogeneous simulations interoperability," in *Proc. of the 2010 Winter Simulation Conference (WSC)*, Baltimore, MD, 2010.

[11] J. Webber, S. Parastatidis and I. Robinson, REST in Practice, O'Reilly Media, 2010.

[12] C. Riva and M. Laitkorpi, "Designing Web-Based Mobile Services with REST," in *Service-Oriented Computing - ICSOC 2007 Workshops. LNCS*, Berlin / Heidelberg, 2009.

[13] G. Wainer, Discrete-Event Modeling and Simulation: A Practitioner's Approach, P. Mosterman, Ed., Taylor and Francis, 2009.

[14] G. D'Angelo, "Parallel and distributed simulation from many cores to the public cloud," in *Proc. of 2011 Int. Conf. on High Perf. Comp. and Simulation (HPCS)*, Istanbul, 2011.

[15] A. Wytzisk, I. Simonis and U. Raape, "Integration of HLA Simulation Models into a Standardized Web Service World," in *Proc. of the Euro. Sim. Interop. Workshop (SISO)*, 2003.

[16] I. Holmes and R. Kawalsky, "The RealityGrid PDA and Smartphone clients: Developing effective handheld user interfaces for e-Science," in *Proc. of UK e-Science*, 2006.

[17] D. Huynh, D. Knezevic, J. Peterson and A. Patera, "High-fidelity real-time simulation on deployed platforms," *Computers & Fluids,* vol. 43, pp. Pages 74-81, April 2011.

[18] P. Longley, Geographic information systems and science, Wiley, 2005.

[19] M. Neteler and H. Mitasova, Open source GIS: a GRASS GIS approach, vol. 689, Kluwer Academic Publishers, 2002.

[20] J. Brauner, "Providing GRASS with a Web Service Interface," in *Proc. of the 6th Geog. Inf. Days*, Münster,2008.

[21] Z. Mednieks, L. Dornin, M. Nakamura and G. B. Meike, Programming Android, O'Reilly Media, 2011.

[22] G. Wainer, "Applying cell-DEVS methodology for modeling the environment," *Sim.,* vol. 82, no. 10, pp. 635-660, 2006.

[23] H. Wright, R. Crompton, S. Kharche and P. Wenisch, "Steering and visualization: Enabling technologies for computational science," *Future Generation Computer Systems,* vol. 26, no. 3, pp. 506-513, March 2010.