International Journal of Modeling, Simulation,
and Scientific Computing
Vol. 7, No. 1 (2016) 1641002 (38 pages)
© World Scientific Publishing Company
DOI: 10.1142/S1793962316410026

World Scientific
www.worldscientific.com

Modeling and simulation as a service architecture for deploying resources in the Cloud

Sixuan Wang* and Gabriel Wainer†

Department of Systems and Computer Engineering
Carleton University Centre for Visualization and Simulation (*V-Sim*)
Ottawa, Canada, ON K1S-5B6
**swang@sce.carleton.ca*
†gwainer@sce.carleton.ca

Received 25 April 2015
Accepted 20 January 2016
Published 7 March 2016

In recent years, Cloud Computing has become popular to facilitate the use of Modeling and Simulation (M&S) resources. Nevertheless, there are still various issues to solve, including the structure constrain of chosen web service frameworks, the sharing of varied resources in the Cloud, and the difficulties in reproducing experiments. We show a new architecture based on Cloud Computing and new modeling methods to deal with these issues. This layered architecture, called Cloud Architecture for Modeling and Simulation as a Service (CAMSaaS), simplifies the deployment of M&S resources as services in the Cloud. CAMSaaS supports hierarchical resource services, experimental frameworks, scalable infrastructure and makes everything as a service. We deploy varied M&S resources as services in the Cloud, and build a Modeling and Simulation as a Service (MSaaS) middleware called CloudRISE to manage a variety of M&S resources. We also use the experimental framework concept to simplify the management of experiment environments. We present a case study for crowd evacuation application using the architecture.

Keywords: Modeling and Simulation as a Service; Cloud-based Simulation; Simulation Middleware; RESTful Web Services; Cloud Services.

1. Introduction

At present, Modeling and Simulation (M&S) is being applied to almost every aspect of life.[1] Current M&S systems are large and complex, and it is thus complex to configure and maintain them in order to run varied experiments of the models under different scenarios. Likewise, the CPU and memory resources of a single computer can be insufficient to execute complex models for advanced applications.[2]

The M&S community has used Web Services (WS) technologies to deal with these issues for around 20 years. This method, usually called *Web-based Simulation*

*Corresponding author.

(WBS), uses existing simulation functions and it exposes them as WSs.[3] WBS shares the simulators and their environments (which are originally accessible on a single computer) through the Internet. This method has been successful, and a large number of M&S WS exist.[4]

The emergence of Cloud Computing made Cloud-based Simulation (CBS) a feasible and attractive alternative to WBS.[5] CBS goes beyond WBS, and it uses Cloud Computing technologies to reduce costs and make easier to develop M&S systems by taking advantages of the Cloud Computing services, e.g., Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). Cloud Computing provides various IT resources to users in an easy-to-use, on-demand and pay-as-you-go manner. These advantages make simulation resources more accessible to users. Nevertheless, research in CBS is still in a preliminary stage; in fact, CBS has been mentioned as one of the Grand Challenges in M&S by various experts in the area.[6–8]

In this sense, Johnson and Tolk[9] have identified five challenges perspectives that have to be addressed in CBS: technical, governance, security, business and conceptual perspectives. Among them, we are interested in focusing on the technical perspective, a fundamental concept, and we focus on how to apply the best technological solutions to deploy M&S resources as services in the Cloud. Although Cayirci[10,11] shows that the technical perspective has been improved, Liu *et al.*[12] argue that the current technical solutions are not practical, because CBS has issues in the following aspects:

(1) WS Framework: CBS is derived from WBS. After a peak period between 2000 and 2002, the research and publications about WBS dropped quickly,[13] because there was a mismatch between the WS Framework used by WBS and the main characteristics of the Web. Most WBS frameworks are based on SOAP, which has issues of structure constrains (e.g., mixed design and implementation, difficulties in development, exposing internal implementation details, etc.).[14] Any SOAP-based WS project takes at least two years in defining the SOAP WS layer, deploying those services, and standardizing the interfaces. Therefore, these SOAP-based WS failed to take full advantage of features of the Web (e.g., its universal interface, interoperability, ease of navigation and use, etc.). In other words, the focus of many WBS efforts has been merely to re-implement existing standalone M&S systems by using SOAP-based WS.[13]

(2) Variety: As the M&S systems become more complex, the number of the related resources increases very fast.[15] There is a variety of basic M&S resources: source systems, models, simulators, experimental frames and experiments.[16] However, as the simulation systems become more complex, new resources are needed.[17] In particular, numerous WS available in the Cloud could be useful for M&S.[4] These services are either web-based M&S services (that expose models or simulations as services) or open APIs (that are useful for M&S). These resources, which are varied in type and content, are complex to share and reuse.

(3) Simplicity: Although CBS is available to a large number of users, running a simulation in the Cloud is still complicated.[18] To run a simulation model from others, a user needs to find right WS to install a simulation environment, which can be complex (it needs to deal with dependencies and configuring the simulation environment). Then one must have access to the model files, inputs, and experiments process. Both current WBS and CBS are still unable to simplify these processes.[3,18]

This paper focuses on the issues mentioned above and introduces new methods for deploying M&S resources in the Cloud. The goal is to simplify the deployment process of varied M&S resources as services in the Cloud by taking full advantages of Web.

To achieve this goal, new concepts and techniques are needed. We propose using RESTful WS[19] as the *WS framework*. REST is simpler than SOAP; it is directly built upon the Web and it has a more concise style in terms of implementation and operation.[20] We use RESTful WS to expose every M&S resource as a unique URL that can be operated by the uniform HTTP methods (GET, POST, PUT and DELETE).

In order to handle the *variety* of M&S resources, we use the concept of Modeling and Simulation as a Service (MSaaS) in the Cloud. MSaaS builds on Cloud services and it delivers WS related to M&S. We implemented an MSaaS middleware to manage and deliver varied M&S resources as services in the Cloud.

In order to improve the *simplicity*, we propose using the Experimental Framework concept, which captures the set of conditions under which a system or a model is to be observed and studied.[16,21] We developed a lightweight Experimental Framework Template using XML in the MSaaS middleware to control the life cycle of experiments. Likewise, we used the Cloud as the underlying infrastructure to deploy the MSaaS middleware.

Based on these new concepts and methods, we propose a CBS architecture to deploy resources as services, named Cloud Architecture for Modeling and Simulation as a Service (CAMSaaS). CAMSaaS is a layered architecture that allows users to deploy and invoke varied M&S resources as services. CAMSaaS uses three layers, namely Cloud, MSaaS, and Application. We developed middleware, named CloudRISE, which implements the CAMSaaS architecture and allows deploying MSaaS in the Cloud. We will introduce the overall CAMSaaS architecture, and its basic ideas of design and implementation.

In order to show the uses of the proposed architecture and the current implementation, we present a case study for evacuation of a crowd in a shopping mall. We discuss the M&S resources needed (e.g., evacuation model, simulator, data collection and 3D visualization), and how to deploy them as service in the Cloud using CloudRISE, and then we will show how to use this generated MSaaS to reproduce different experiments in an easy and scalable way.

This paper is organized as follows. In Sec. 2, we review the literature related to the deployment of M&S resources as services in the Cloud. In Sec. 3, we discuss our view about MSaaS and propose the new architecture CAMSaaS. In Sec. 4, we discuss the details about the underlying Cloud layer of CAMSaaS. In Sec. 5, we discuss the details of CloudRISE middleware, which implements the MSaaS idea for CAMSaaS. In Sec. 6, we present a complete case study of Crowd M&S using CloudRISE middleware. In Sec. 7, we will conclude this research by highlighting its advantages and future directions.

2. Related Work

We are interested to improve the deployment of M&S resources as services in the Cloud. To do so, we have first classified and organized the variety of M&S resources in order to make them more manageable. Figure 1 shows how we can group the varied resources available.

As we can see, an M&S resource can be considered as an entity (i.e., a basic resource that is directly related to M&S) or a supported resource (i.e., a related resource that is helpful for M&S).

The theory of M&S presented in Ref. 16 provides a conceptual framework for M&S *entities*. M&S entities include the *source system*, *models*, *simulators*, *experimental frameworks* and *experiments* (seen in the left part of Fig. 1). The source system is a real or virtual environment that we are interested to model. A model is a representation of a source system built to understand that system under a given experimental framework. A simulator is a device that executes a model in order to study its behavior over time. The simulation of a model involves different simulation experimental frameworks and associated experiments. The experimental framework represents the context under which a system or a model is observed or experimented with. This kind of context is essential information to reproduce the simulation experiments.

The *supported resources* can be grouped into two categories: supported *data* and supported *functions*, as seen on the right part of Fig. 1. A supported data can be any resource that is not executable, like a system behavior database, scenario data and documentation. Supported data can be in a wide range of forms: text, file, picture and video.[17] Supported functions are any resources executable from small

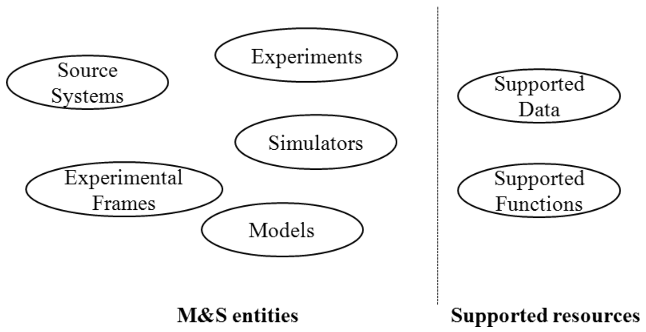


Fig. 1. Categories of M&S resources: basic M&S entities and supported resources.

portions of code, through larger components, to complete executable programs.[22] Examples of the supported functions are data collection tools, results analysis program, animation and visualization tools. A function is in general associated with an execution environment and provides interfaces for others to access.

In order to facilitate the use of varied M&S resources, the M&S community has been interested in web technologies for many years. These efforts include WBS (exposing M&S functions as WS) and CBS (integrating WBS and Cloud Computing). The basic idea of WBS is to put M&S resources on the Web, and to provide users with new information standards and communication protocols.[23] As reported in Ref. 24, web-based technologies are beneficial to non-experts, who can share M&S resources and reduce the time required for users to learn about the M&S tools. In recent years, CBS has started to become popular to facilitate the use of these varied M&S resources.[25] CBS and WBS have some similarities. CBS is derived from WBS and it has been identified as a challenge in M&S.[7,8] However, the work on WBS is not new. The early efforts in using WBS to support modeling, simulation and simulation results analysis can be traced back to Fishwick's paper in 1996. Between 1996 and 2000, the number of publications on WBS grew explosively. After 2002, the number of publications dropped off very quickly.[5] Since WBS and CBS are similar, it is important to study what has happened to WBS.

In WBS, the simulator and its simulation environment are located remotely on a server.[26] Users can submit their requests (with specified message/parameters) to the simulator through web servers, then simulation experiment runs remotely, and the results are returned to the user. WBS eases the sharing of the simulation resources: for instance, the simulation engines available on the server (without worrying about simulation environment setup and other software dependencies issues). WBS improves data accessibility, interoperability and user experience (non-expert users can access these services easily), allowing them increase productivity.[27]

Onggo and Taylor[5] analyzed the problem of the decline of WBS. They stated that there was a mismatch between the main characteristics of the Web and the web service framework taken by the domain of WBS (which mainly uses SOAP-based WS). This mismatch resulted in that WBS cannot take full advantage of the features of the Web including common standards, interoperability, ease of navigation and use, etc. Kuljis and Paul[28] also supported this argument. They argued that CBS must not simply re-implement existing simulation software using Cloud Computing technologies.

WS can be categorized into two main frameworks: SOAP-based[29] (in which the service expose an arbitrary set of operations), and RESTful[30] (in which we manipulate XML representations of Web resources using a uniform set of "stateless" operations). In WBS, SOAP-based WS are widely used in M&S systems, while few systems use REST-based WS as the web service framework.

SOAP-based WS are exposed as in Remote Procedure Calls (RPC). For simulation, the simulation functions are available as procedures on the server, and they are described in XML WSDL documents. A client can compile the WSDL into procedure stubs, and at runtime, the SOAP messages are wrapped in HTTP, and POSTs the SOAP message to the server. When the server receives the message, it extracts it using a procedure call and responds to the client in a similar way. DDSOS[31] is a simulation framework by using SOAP-based WS to support the evaluation of large-scale distributed systems. SOPM[32] is a SOAP-based modeling framework to build performance models of service-oriented architectures. SASF[25] is a SOAP-based simulation framework for predicting the behavior of service-oriented systems under different configurations and loads. There are many SOAP-based WS used in DEVS simulators. In Refs. 33–35, we defined a distributed DEVS simulation environment over SOAP. DEVS/SOA[36,37] implements DEVS over a SOAP-based SOA framework, supporting a development and testing environment known as DEVS Unified Process. A similar work was proposed in SOA-compliant DEVS (SOAD), a simulation framework for modeling service-oriented computing systems.[38] They developed a set of abstract DEVS models that conform to the SOA principles.

Instead, RESTful WS imitate the Web interoperability style, directly take full advantage of the Web. They use universal accepted standards, uniform channels, are resource-oriented, message-oriented, and hide implementation. REST represents resources as URIs; any user can request the resource by HTTP method, and communicate with the resource via standard representation (like XML). Resources are manipulated using a fixed set of four operations: POST, GET, PUT, and DELETE. However, using REST for WBS is not as popular as SOAP. The RESTful Interoperability Simulation Environment (RISE)[39] was the first RESTful distributed DEVS simulator in the Cloud. The main objective of RISE is to support interoperability of distributed and heterogeneous simulations regardless of the model formalisms, model languages or simulation engines. Arroqui *et al.*[40] developed an agricultural information system by using both RESTful WS and SOAP-based WS. From the comparison, they found that RESTful WS requires 24% less bandwidth for transferring data than SOAP-based WS do.

SOAP WS have the following shortcomings when compared to RESTful WS:

(1) Client–Server interaction in SOAP WS is tightly coupled, and changes on the server result in complex code changes on the client.[14,39]
(2) SOAP messages require XML wrappers on all messages, whereas REST does not. Therefore, SOAP messages are bigger than HTTP messages used in RESTful WS; hence, SOAP consumes more bandwidth.[14,41]
(3) SOAP exposes details of the internal implementation. Developing SOAP WS needs the understanding of the detail implementation of the procedures, which makes them harder to develop.[14,42]
(4) SOAP is based on RPC. Each RPC interface defines its own services with its own syntax and semantics, which limits the scalability of the framework; while REST has a uniform interface because of its use of HTTP methods.[43,42]
(5) SOAP WS also has security issue. SOAP/RPC messages are wrapped in an HTTP header; the server only understands the messages once they have been

parsed. This allows malicious commands to get through the firewall undetected. In contrast, each resource in REST is assigned a unique URI, and it is handled on the HTTP firewall (making REST more secure).[44]

In fact, a large number of RESTful WS have been designed as replace inefficient SOAP WS. According to more than 11,000 APIs registered by Programmable Web, the trend toward SOAP is dying: today 73% of the APIs use REST, while SOAP is only 27%.[45] These new services in REST are more scalable, interoperable and simpler.[43] These shortcomings of SOAP mentioned above have made WBS unable to take full advantages of Web, which further made WBS decline in recent years. Therefore, in CBS, RESTful WS should be adopted as Web Service framework instead of SOAP WS.

CBS is a term used to capture the intersection between M&S and Cloud Computing. Cloud Computing can be used for building simulation environment in a pay-as-you-go manner, reducing the costs in the development of M&S applications,[11] and computing power can be increased or decreased according to usage. Cloud computing also allows the users to scale up or down the underlying infrastructure.[12] We can use Cloud Computing to manage varied M&S resources and build different simulation environment according to the actual demands. Cloud Computing for WBS also uses advanced technologies such as load balancing, fault tolerance and security.[46] In Ref. 5, the authors have pointed out the following Cloud-based scenarios for M&S: (1) run simulation on Cloud infrastructure; (2) create and control models in the Cloud; (3) run experiments of simulation models on Cloud infrastructure; (4) manage and control the simulation development lifecycle; and (5) control over storage and execution simulation engine.

Johnson and Tolk[9] identified five challenges that have to be addressed in CBS (technical, governance, security, business and conceptual). Cayirci[10,11] showed how various technical perspectives have been addressed, but Liu *et al.*[12] argue that the current technical solutions are neither widely used nor practical. Onggo and Taylor[5] also argued that deploying M&S resources in the Cloud is not simply as the way WBS does: reimplementation of existing M&S resources using Cloud Computing.

In the following sections we will show how Cloud Computing can help us to develop MSaaS, a concept that received some attention recently. MSaaS focuses on delivering services related to M&S using Cloud Computing,[10] exposing models and simulation functions as services. MSaaS takes advantages of IaaS, PaaS and SaaS,[47] hiding the underlying infrastructure, platform and software details from the users.

Recent research focused on how to use Cloud and MSaaS for reusing M&S resources. There are three groups of researchers, which focus on different aspects:

(1) Managing M&S resources: They use the Cloud to share and manage M&S resources. Sliman *et al.*[48] proposed an MSaaS platform called RunMyCode, which allows scientists to share their code associated with their research publications in the Cloud. In Ref. 12, the authors presented the work on deploying existing M&S software into the Cloud.

(2) Supporting Parallel and Distributed Simulation (PADS): Li *et al.*[49] proposed a Grid-based platform called Cloud Simulation Platform (Cesium), and summarized 12 key technologies for the development of CSim. Fujimoto *et al.*[46] discussed the benefits and challenges associated with executing PADS in the Cloud.

(3) Building applications in the Cloud: They focus on the development of M&S applications using the Cloud. Bruneliere *et al.* presented an approach to build applications by using model-driven engineering and Modeling as a Service (MaaS) in the Cloud. Lanner group[51] designed a system called L-SIM 2.0 in the Cloud to simulate business process management systems. CIMdata[52] proposed a simulation application using the Cloud in the area of commercial product design.

The use of Cloud Computing and MSaaS is still in a preliminary stage and a universal definition of MSaaS does not exist.[11,53] In Ref. 11, the author discussed the differences and relations among MSaaS, IaaS, PaaS and SaaS. In particular, the author views MSaaS as a special form of SaaS. The author classified three types of MSaaS: modeling as a service (services to develop models), model as a service (services to access models) and simulation as a service (services to run simulations). However, this kind of classification of MSaaS only considers models and simulations; it does not consider other kinds of M&S resources (e.g., sharing models and experiments, storing supported data and reproducing supported functions). For example, in order to complete a simulation process, some supported functions are important (e.g., data collection for the model inputs, simulation results analysis). However, the current WBS and CBS lack the support for users to expose these supported functions and reproduce the experiments related to these functions.

Our research effort can also help in dealing with the reproducibility crisis: in Ref. 54, the author analyzed many papers from the ACM MobiHoc symposium between 2000 and 2005, and found that less than 15% of the simulation results were reproducible. There are several reasons for this. Then one must reproduce the simulation in the same way as the original, which requires that one must have the access of simulation services for the same simulation files, inputs, etc. In this sense, existing WBS and CBS lack in simplicity in terms of environment configuration, which makes reusability very complex. In WBS, it is hard to simplify the simulation process.[3] Most services provided by WBS are delivered on local servers, so these services rely on heavily the capacity of the servers. Due to the increasing demands of underlying infrastructure for simulation experiment, it is hard to set up new servers efficiently with a higher compute power or a larger storage capacity. Users have to migrate and backup by themselves, as well as purchasing infrastructure and IT services. In CBS, even users can use Cloud services to get scalable computer power or storage capacity, carrying out an experiment can be still complicated. User still need to select correct services to set up experiment environment, prepare input, control experiment lifecycle and analysis results.[18] It is even harder when there is a varied M&S resources that cannot be easily exposed as services. Therefore, a simplified way to develop these resources as services in the Cloud is needed.

In summary, though web technologies have been used in M&S for many years, deploying M&S resource in the Cloud should not simply re-implement existing systems in the Cloud; instead, we need to handle the issues caused by web service framework like SOAP WS, the variety of M&S resources and the difficulty of environment configuration. The following sections present a novel architecture to deal with these issues.

3. Cloud Architecture for Modeling and Simulation as a Service

As discussed in the previous sections, we are interested in simplifying the deployment process of varied M&S resources as services in the Cloud. In particular, we want to solve the issues caused by current CBS in the deployment process, including variety, WS frameworks used and simplicity. To do so, we propose using MSaaS to share the varied M&S resources as services. We regard MSaaS in a broad view that it delivers *everything as a service* using RESTful principles. In other words, MSaaS is able to deliver everything related to M&S as RESTful WS by using Cloud Computing. MSaaS is built on top of the Cloud services (i.e., SaaS, PaaS, and IaaS), considering that MSaaS is a special form of SaaS.[11] Figure 2 shows the relationship between MSaaS and the Cloud services.

Users can share a model by using *Model as a Service*, reproduce experiments by using *Experiments as a Service*, reconfigure Experimental Frameworks by using *EF as a Service*, run a simulation by using the corresponding *Simulator as a Service*, share the supported data by using *Data as a Service*, and execute supported functions by using *Function as a Service*. These MSaaS services take advantages of SaaS, PaaS and IaaS. SaaS provides software on-demand and application services to be used for MSaaS. PaaS provides a computing platform that facilitates the development, deployment and management of the applications for MSaaS. IaaS delivers computer and storage IaaS for MSaaS to access.

Based on these ideas, we propose a novel architecture, named CAMSaaS to deploy varied M&S resources as MSaaS in the Cloud. Figure 3 shows its layered architecture. This architecture allows the users to deploy any user-provided

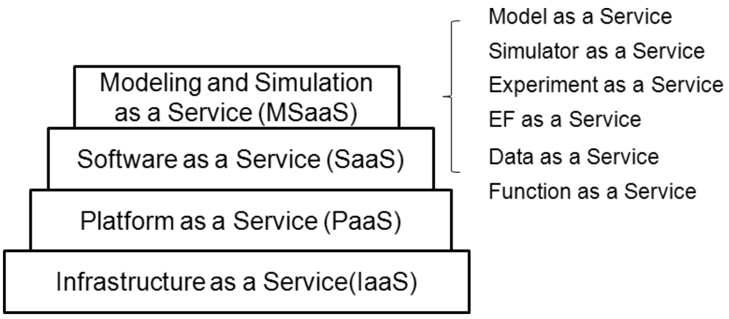

Modeling and Simulation as a Service (MSaaS)
Software as a Service (SaaS)
Platform as a Service (PaaS)
Infrastructure as a Service(IaaS)
Model as a Service
Simulator as a Service
Experiment as a Service
EF as a Service
Data as a Service
Function as a Service

Fig. 2. Relationship of MSaaS and Cloud services: Varied M&S resources exposed as MSaaS.

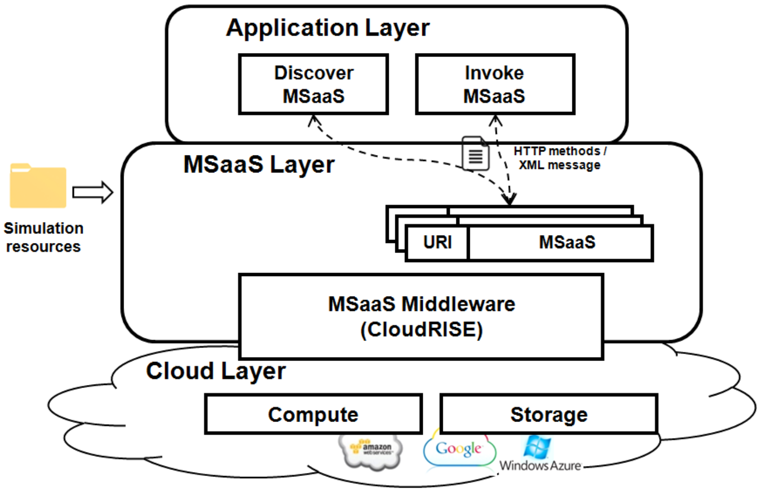


Fig. 3. CAMSaaS. It contains: Cloud Layer (Cloud infrastructure), MSaaS Layer (deploying MSaaS using middleware) and Application Layer (developing new applications using MSaaS).

resources as MSaaS in the Cloud. Users can use these MSaaS to share the M&S resources and reproduce the experiments in an easy way. This architecture has the following three layers.

(1) Cloud Layer: It is responsible for providing Cloud infrastructure support, like Cloud compute units and Cloud Storage Units. This layer supports the scalable underlying infrastructure for M&S resources. All resources will be stored and all the experiments will be executed in this layer. Cloud compute units are used for building simulation environments and executing experiments. Cloud Storage Units are used for sharing M&S resources.

(2) MSaaS Layer: It is responsible to provide a platform for development, deployment and management of MSaaS. This layer can deploy user-provided M&S resources as MSaaS in the Cloud by using the *MSaaS middleware* on-demand. This middleware provides an Experimental Framework Template for controlling the lifecycle of experiments. In addition, this layer can easily launch experiment environment using the services from the Cloud Layer.

(3) Application Layer: It is responsible to develop new applications by using the MSaaS services that are generated from the MSaaS Middleware Layer. Users can invoke these services through URIs by XML messages using HTTP methods.

This architecture supports collaborative development with different kinds of users:

(1) The **Cloud Administrators** manage the Cloud compute units and Cloud Storage Units. They support the management of the underlying infrastructure for the upper layers.

(2) The **M&S Providers** are specialists from different fields who can provide a variety of M&S resources. They use MSaaS middleware to deploy these resources as MSaaS services.
(3) The **Clients** are the end users, who can discover the generated MSaaS services and use the ones that they are interested in their applications.

The CAMSaaS architecture allows each kind of user to focus on their own task, and it makes it easy for the users of different layers communicate and collaborate. At the beginning, it starts with the **Cloud Administrator**. They configure the Cloud infrastructure (e.g., adding basic dependencies and configurations of computing units). They can take advantages of many Cloud services, like the ones provided by Amazon EC2, Google App Engine, and Microsoft Azure. After that, the **M&S Providers** from different domains can deploy their M&S resources as MSaaS in the Cloud. They can use the MSaaS middleware to develop and manage the varied M&S resources dynamically (e.g., models, simulators, experiments, supported functions, supported data, Cloud images and instances). The **Clients** can use these newly deployed MSaaS following RESTful WS principle to reproduce the experiments. They can discover and invoke services from the generated MSaaS in their applications, following RESTful WS framework.

In the following sections, we will introduce the design and implementation details of the lower layers (i.e., the Cloud Layer, the MSaaS Layer and its implementation CloudRISE, and a case study as application by using MSaaS).

4. Cloud Layer

As discussed earlier, an important issue for deploying M&S resources in the Cloud is the difficulty in terms of environment configuration. We need to set up the underlying infrastructure for the M&S resources. In Sec. 3, we have briefly introduced the CAMSaaS architecture. The Cloud Layer is the bottom layer of the CAMSaaS architecture, and it has the goal to deal with scalability by using Cloud Computing. This layer uses the Cloud as the underlying infrastructure, and it can thus provide IaaS services for the upper layers of the CAMSaaS. Physically, all resources will be stored and all the experiments will be executed in this layer. In this section, we will discuss details of the design and implementation of this layer.

Figure 4 shows the basic structure of the Cloud Layer. In general, this layer supports the Cloud infrastructure, and it provides IaaS APIs.

As seen in Fig. 4, we consider two kinds of Cloud infrastructures: *Compute Units* and *Storage Units.* The Compute Units are the fundamental infrastructure for executing experiments (either for simulations or for supported functions), while the Storage Units store M&S resources (e.g., the models, the experiments, the EFs, the supported data, and the supported functions). The *IaaS APIs* (divided in *IaaS compute APIs* and *IaaS storage APIs*) are the APIs of the IaaS services for these Cloud infrastructures. They can manage and access the underlying Cloud infrastructures (e.g., launching/releasing new Compute Units, saving/retrieving resources

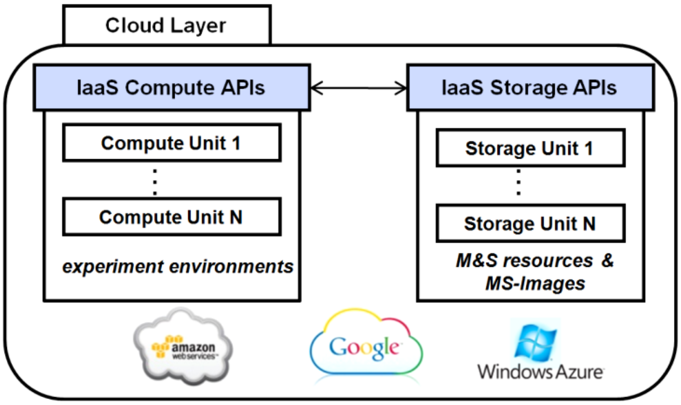


Fig. 4. Basic structure of the Cloud Layer. It has Compute Units (computational infrastructure to run experiments), Storage Units (storage for varied M&S resources) and IaaS APIs (APIs of the IaaS services for the Cloud infrastructures).

from a Storage Unit). In recent years, many Cloud providers made available IaaS services (e.g., Amazon Web Service — AWS, Google Compute Engine, or Microsoft Windows Azure).

The IaaS APIs provide a simplified version of these services, which facilitate the upper layers of CAMSaaS to access the Cloud. In particular, the MSaaS middleware in the Component Layer (built for deploying M&S resources) can use these IaaS APIs to scale up and down the number of the Compute and Storage Units.

For the Compute Units, they have compute power for computing needs. They can be physically located in a large number of computers, which can be distributed across data centers. The Compute Units in CAMSaaS contain *experiment environments* (runtime environments for the simulations or supported functions), which can use different environments for different simulators and functions. Each Compute Unit can access assigned experiments from a Storage Unit, and reproduce it in the corresponding experiment environment. After the experiment finishes, the Compute Unit saves the experiment's outputs back to the storage.

For the Storage Units, each M&S resource can be stored in various physical storage volumes distributed and connected in the Cloud. These Storage Units can be automatically replicated to protect the data from failure, providing high availability and accessibility. For instance, we can access any M&S resource stored in a Storage Unit at any time from any place as long as we have Internet access.

There are numerous IaaS services to access the Cloud infrastructure, offered by different providers on-demand. In order to use them, the users need to choose the right ones and configure them. For instance, they need to specify the IaaS providers, deal with the credentials, configure the security group, and set up the connection with the Cloud. In order to simplify this process and to provide the upper layers an easy way to use these services, we developed the *IaaS APIs*. These include the *IaaS Compute APIs* (to manage the Compute Units) and *IaaS Storage APIs* (to manage the Storage Units). More details of the IaaS APIs are discussed later in this section.

In order to provide a simple and scalable way to launch experiment environments for M&S resources, we propose the concept of the *MS-Image*, a copy of Compute Unit consisting of the MSaaS middleware, the simulators and the environments for supported functions (Fig. 5). An MS-Image is initially saved in a Storage Unit, and it can be launched as any number of Compute Units. Different MS-Images can use different kinds of simulators and function environments. The concept of MS-Image is related to the Virtual Machine (VM) images in the Cloud. In fact, an MS-Image is a special version of VM image. A VM image is a virtual server image that includes an operating system and cloud-related software. For instance, in AWS there are VM images in different categories, such as operating system (e.g., Linux or Windows), architecture (e.g., 32-bit or 64-bit) and region (e.g., public or private). Users can launch Compute Units from a particular VM image. The main difference between an MS-Image and a VM image is that the MS-Image can also be viewed as a customized VM in the Cloud. An MS-Image contains not only an operating system, but also MSaaS middleware, simulators and function environments. An MS-Image takes advantages of the VM image, and it is built for launching an M&S environment quickly.

As seen in Fig. 5, an MS-Image consists of:

(1) MSaaS middleware: It uses the IaaS APIs to deploy the varied M&S resources. It separates the underlying infrastructure and the MSaaS services, wrapping the simulators and functions provided by users and exposing the experiment environments for them.

(2) Simulators: The MS-Images configure the simulation environments for different simulators. Different MS-Images can configure different simulators and

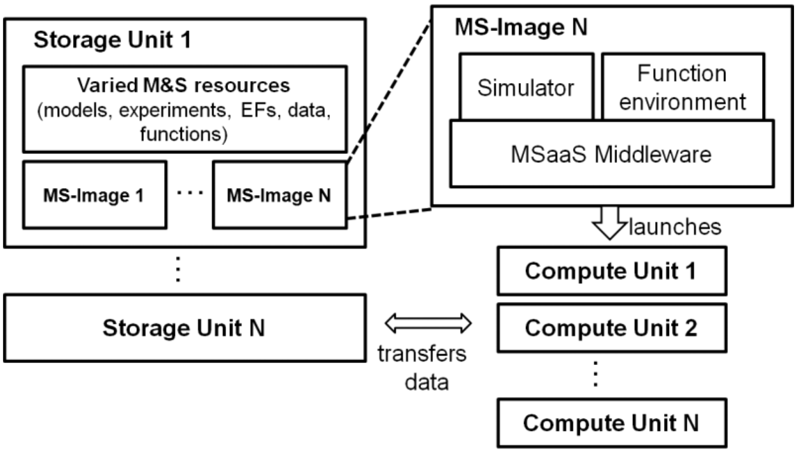


Fig. 5. MS-Image with MSaaS middleware, simulator, and function environment.

allow them to be ready to use by the MSaaS middleware. Users can use the MSaaS services provided by the MSaaS middleware to find the needed simulator and reproduce experiments on it.

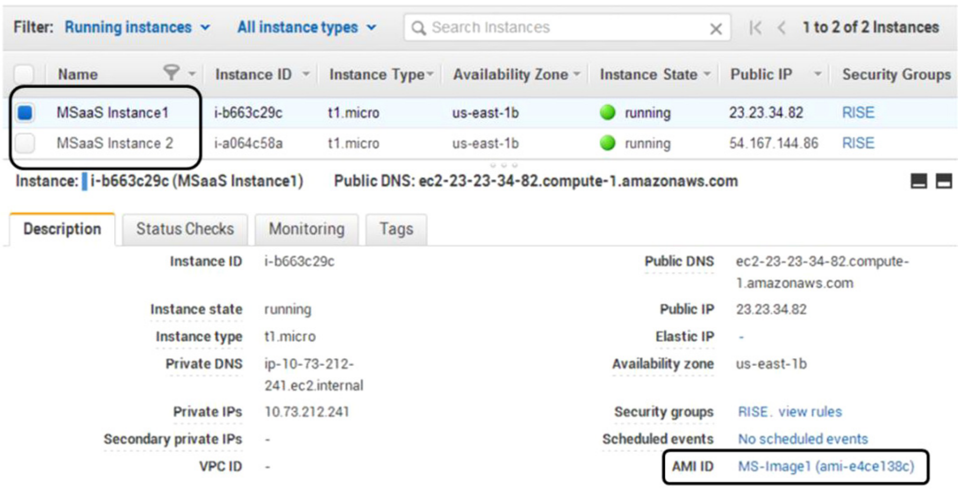
(3) Environments for supported function: MS-Images can hold environments for different supported functions (e.g., Java, C++, and Python). Users can deploy their own functions on these environments at run-time, using the MSaaS services to reproduce experiments.

The MS-Image can scale up/down the experiment environments by launching/releasing Compute Units. Users do not need to worry about the installation and configuration of the experiment environments. If they want to reproduce experiments, they can find an MS-Image with the simulator or the function environment needed, and launch Compute Units with that image. After that, they can use the MSaaS middleware to reproduce the experiments. For instance, a simulation model needs to simulate 50 times. We can find an MS-image that contains the needed simulator, and launch 10 Compute Units of that image. After that, we can execute five simulations on each Compute Unit. When the simulations complete, we can release these 10 Compute Units to reduce costs.

In CAMSaaS, the Cloud administrators are responsible to manage the underlying Cloud infrastructures. They can use the IaaS APIs and MS-Images to launch or release the infrastructure. In addition, they can also use other types of Cloud services (e.g., PaaS and SaaS) to manage the Cloud infrastructures better. For instance, the PaaS load balancing service can distribute messages among Compute Units evenly. An example of SaaS is Amazon Simple Notification Service, which can send notifications via email when infrastructure health changes. Other CAMSaaS users (i.e., M&S providers and Clients) do not operate this layer; instead, they use the MSaaS middleware.

In the current implementation, we developed a prototype of the Cloud Layer using AWS, a popular Cloud provider that offers IaaS services charged hourly or monthly. Please note there are also other Cloud providers (like Google Compute Engine, or Microsoft Windows Azure), the reason we chose AWS as the Cloud provider is that AWS supports lower level operations compared to other providers; so one can completely control the underlying infrastructure and build the MSaaS middleware using AWS. In particular, we used Amazon Elastic Compute Cloud Instances (*Amazon EC2 Instances*) for Compute Units, *Amazon S3* for Storage Units, *Amazon Machine Images* (*AMI*) for MS-Image, and *AWS SDK* for IaaS. Amazon EC2 Instances provide resizable compute capacity in the Cloud. Amazon S3 is a Cloud storage service. AMI allows launching Amazon EC2 Instances, and AWS SDK provides IaaS with APIs for many AWS services (e.g., Amazon S3, Amazon EC2).

We can launch EC2 Instances implementing the Compute Units by specifying their *instance type* and *AMI*. Instance types have different combinations of CPU/GPU, memory, storage, and networking capacities. For example, *Amazon*



Filter: Running instances · All instance types · Search Instances · 1 to 2 of 2 Instances

Name	Instance ID	Instance Type	Availability Zone	Instance State	Public IP	Security Groups
MSaaS Instance1	i-b663c29c	t1.micro	us-east-1b	running	23.23.34.82	RISE
MSaaS Instance 2	i-a064c58a	t1.micro	us-east-1b	running	54.167.144.86	RISE

Instance: i-b663c29c (MSaaS Instance1) Public DNS: ec2-23-23-34-82.compute-1.amazonaws.com

Description · Status Checks · Monitoring · Tags

Instance ID	i-b663c29c	Public DNS	ec2-23-23-34-82.compute-1.amazonaws.com
Instance state	running	Public IP	23.23.34.82
Instance type	t1.micro	Elastic IP	-
Private DNS	ip-10-73-212-241.ec2.internal	Availability zone	us-east-1b
Private IPs	10.73.212.241	Security groups	RISE. view rules
Secondary private IPs	-	Scheduled events	No scheduled events
VPC ID	-	AMI ID	MS-Image1 (ami-e4ce138c)

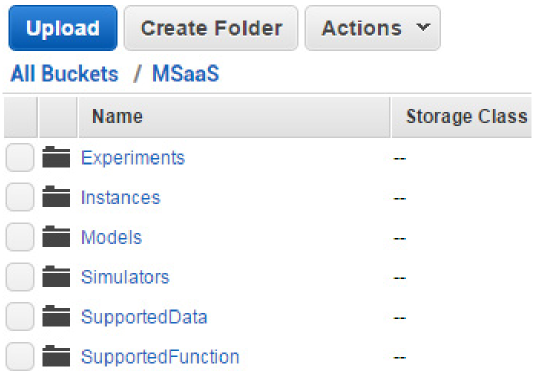
Fig. 6. Amazon EC2 instances for CAMSaaS. Two instances (MSaaS Instance 1 and 2) launched from a same customized AMI (MS-Image1).

Micro is low-cost, providing a small amount of CPU. In our implementation, the AMI is customized to implement MS-Images: we launch different EC2 instances from an AMI according to the actual needs. The customized AMI contains CloudRISE (the implementation of MSaaS middleware) and supports different experiment environments for three different versions of CD++. We can also add other simulators in the customized AMI. For the function experiments, the current customized AMI uses Amazon Linux AMI 2013, which covers environments for most functions (e.g., Java, C++, Python, PHP, and Tomcat). If a function needs a special environment, we can add on the AMI with additional utilities and services.

The Cloud administrator can use the AWS Management Console to manage Amazon EC2 instances. Figure 6 shows a screenshot of the current implementation.

We have two instances (shown in the top-left of Fig. 6): *MSaaS Instance 1* and *2*, each with its own IP. Each instance is launched by the same customized AMI (*MS-Image1*), shown in the bottom-right of Fig. 6, which contains the MSaaS middleware, two version of CD++ simulators (CD++ and CD++3.0),[55] and the basic function environments (supported by AMI Linux 2013).

We use Amazon S3 to store varied M&S resources. Amazon S3 provides IaaS services that can be used to store and retrieve any amount of data, at any time, from anywhere on the web. We choose Amazon S3 because it is simple and highly scalable. Amazon S3 can scale up its storage capacity automatically based on the current needs. It stores resources as objects within buckets (a container to hold objects). Amazon S3 can be seen as a Cloud-based file system, in which each object is uniquely identified by a URL. For example, in a URL like *http://s3.amazonaws.com/MSaaS/Instance/.../ms-image1.ami*, *MSaaS* is the name of the bucket and *Instance/.../ms-image1.ami* is the key of the object. In the



Name	Storage Class
Experiments	--
Instances	--
Models	--
Simulators	--
SupportedData	--
SupportedFunction	--

Fig. 7. Amazon S3 resource folders for CAMSaaS. They are separated by the categories of the resources (Models, Simulators, Experiments, Instances, Supported Data and Supported Function).

current implementation, we use one bucket for all the varied M&S resources, named *MSaaS* (here we use one bucket since it is easier to manage; but the number of buckets does not matter, we can also separate objects in different buckets). The structure of the objects in the bucket is hierarchical. The first level of the folders are */Model, /Simulator, /Experiments, /SupportedData, /SupportedFunctions,* and */Instance,* which correspond to the varied M&S resources and the MS-Images respectively. Figure 7 shows the current resource folders by using the AWS Management Console. Each folder can have subfolders to storage a given type of resource files.

The Cloud Layer is responsible to provide the IaaS APIs for the MSaaS middleware to access the Cloud. As discussed earlier, these IaaS APIs are built on the IaaS services, and they expose a simplified APIs (e.g., easy configuration). In the current implementation, we use AWS SDK as the IaaS services. We implemented IaaS APIs for the AWS SDK in an easy way. For example, we implemented the IaaS Compute APIs in a utility class (*AWSEC2Instances.java*) and the IaaS Storage APIs in a utility class (*AWSS3Storage.java*).

The class *AWSEC2Instances.java* provides IaaS Compute APIs for the MSaaS middleware to work with AMI and EC2 instances. Table 1 shows some of the APIs.

Table 1. IaaS Compute APIs: AWSEC2Instances.java for Amazon AMI and EC2.

IaaS Compute APIs	Description
boolean setUpConnection (String accessKey, String secretKey, string securityGroup)	Sets up a connection to the EC2 instances.
boolean createImage (String instanceId, String ImageName)	Creates a customized AMI from an existing instance that is either running or stopped.
boolean runInstances (String imageId, StringinstanceType, int count)	Launches specified number of instances using an AMI
boolean terminateInstances (ArrayList instanceIDs)	Shuts down one or more instances.

Table 2. IaaS Storage APIs: AWSS3Storage.java for AmazonS3.

IaaS Storage APIs	Descriptions
boolean setUpConnection(String accessKey, String secretKey, string securityGroup)	Sets up a connection to the MSaaS S3 repository.
boolean createResource(String resourceType, String resourceName, File file)	Creates a new resource to the MSaaS S3 repository.
boolean updateResource(String resourceType, String resourceName, File file)	Updates an existing resource in the MSaaS S3 repository.
boolean deleteResource (String resourceType, String resourceName)	Deletes an existing resource repository.
boolean downloadResource(String resourceType, String resourceName, FileoutputFile)	Downloads a resource from S3.

It supports the functions to set up the configuration with the Cloud (*setUpConnection*), to create an existing instance as a customized AMI (*createImage*), to launch new EC2 instances (*runInstances*), and to release existing EC2 instances in EC2 (*terminateInstances*). For example, when we want to create an instance from a given image, we can call the function *runInstances(String imageId, String instanceType, int count)*, which launches the instances with the request of the AMI (*imageId*), the instance type (*instanceType*), and the number of the instances (*count*).

The class *AWSS3Storage.java* provides IaaS Storage APIs (see Table 2). It supports the functions to set up the configuration with the Cloud (*setUp-Connection*), and to create/update/delete/download (*createResource/delete Resource/deleteResource/downloadResource*) the varied M&S resources in S3. For example, when a new model */evacuation.ma* needs to be saved in AmazonS3, we use *createResource(String resourceType, String resourceName, File file)*, which creates a new resource with the request of Models (*resourceType*), model name (*resourceName*), and the model file (*file*). After that, this model is created in Amazon S3 and can be downloaded in *https://s3.amazonaws.com/MSaaS/Models/Cell-DEVS/Evacuation/evacuation.ma*.

Please note that the MS-Image concept in CAMSaaS is similar to the *DEVS VM kernel* proposed by Mittal and Risco-Martin.[37] It is a DEVS kernel embedded in a VM that can be executed in local, distributed and real-time environments. The MS-Image and DEVS VM kernel have some similarities: they provide VM-like images for simulators and they can be viewed as customized VM for M&S. However, there are many differences between them, listed in Table 3.

The MS-Image differs from DEVS VM kernel in the following aspects:

(1) The DEVS VM kernel aims to provide a general DEVS simulation environment while the MS-Image also supports varied functions and middleware in the Cloud. The MS-Image does not only focus on simulators, while the DEVS VM kernel is dedicated to DEVS simulator.
(2) The DEVS VM kernel is deployed on an environment that can be local, distributed or real-time while the MS-Image is designed for the Cloud.

Table 3. DEVS VM kernel versus MS-Image.

Aspect	DEVS VM kernel	MS-Image
Content	DEVS simulator environment	Environment for DEVS simulators, supported function and Cloud middleware
On-promise deployment	Yes	Potential
Implemented with Cloud Provider	No	Yes (AWS)
Simplicity of deployment on Cloud	Unknown	Single API call by MSaaS middleware
Overall controlled	Unknown	MSaaS middleware

(3) The DEVS VM kernel provides DEVS simulation protocol and it is originally designed for WBS. As mentioned in Ref. 13, it has the potential to be used in a Cloud environment; however, at this time there is no implementation of DEVS VM in the Cloud. In contrast, MS-image is designed for Cloud deployment and it is already implemented using AWS.

(4) Since there is no actual implementation, it is unknown how easy to deploy DEVS VM kernel in the Cloud. For MS-Image, it is simple. MS-Image is highly available and accessible through AWS S3 Cloud Storage, and we can use Middleware to use only single API call for launching Compute Units. The MS-Image is self-configured when launching, without user intervention. After launching, MS-Image automatically deploys simulators and function's environments, installs the packages and dependencies needed, sets up the middleware workspace, and starts the MSaaS middleware service.

(5) The MS-Image is fully controlled by the MSaaS middleware. The MSaaS middleware only uses RESTful WS to control M&S resources. Each launched Compute Unit has a running MSaaS middleware. The MSaaS middleware acts as a VM manager, which can manage and control all the M&S resources including the Cloud infrastructure (e.g., Compute Units, MS-Images). This kind of MSaaS middleware is not supported by DEVS VM kernel.

The Cloud Layer focuses on providing IaaS services with scalable Cloud infrastructure for the CAMSaaS architecture. In the next section, we will discuss the details of the MSaaS middleware that uses the IaaS services provided by the Cloud Layer.

5. Implementing an MSaaS: The CloudRISE Middleware

As discussed earlier, the CAMSaaS architecture has three basic issues: **simplicity** in terms of environment configuration and the MSaaS middleware deals with the structure constrains caused by the **web service framework** (i.e., SOAP WS) and the **variety** of resources. In the current implementation, we implemented the concept of MSaaS in the CloudRISE middleware. CloudRISE is responsible for deploying varied M&S resources as services in the Cloud, using the IaaS

APIs provided by the Cloud Layer. In this section, we will describe the details of CloudRISE.

CloudRISE is an extended version of RISE.[39] Originally, RISE exposes the simulation functions as RESTful WS. The main objective of RISE is to support interoperability of distributed simulations. However, RISE has some restrictions. It was not designed to support the reuse of varied resources in the Cloud. RISE supports an XML configuration file with specific information for each simulation; however, this XML file is specific for DEVS models, and it does not provide general information for the models context (e.g., initial state, termination conditions, inputs/output variables, objectives, assumptions, or model constraints). CloudRISE reuses RISE basic functions (i.e., distributed simulation) and deals with the issues above. It uses Cloud services to share varied M&S resources and reproduce experiments (for either the simulations or functions), and improves reproducibility, using a template for Experimental Frames, which cover the context of a model to reproduce experiments and supported functions.

CloudRISE uses a resource-oriented design based on RESTful WS. M&S resources are identified through URIs, and they exchange information through standard XML messages via HTTP methods (GET/PUT/POST/DELETE). As we saw in the Cloud Layer, all resources are stored in Storage Units (e.g., AWS S3). CloudRISE works as a repository interface for users to share and manage the M&S resources. More importantly, each Compute Unit (e.g., EC2 instances) has an associated MSaaS middleware (i.e., CloudRISE) when it is launched from a MS-Image. CloudRISE lists all the resources available in the Storage Units and the environments available for experiments in the Compute Units. For instance, each URI starts with the endpoint like *http://www.amazon-ec2ip.com:8080/msaas/...*, which indicates the public IP and port of an EC2 instance on which a CloudRISE is running. Figure 8 shows how the varied M&S resources are organized by using the URI hierarchy.

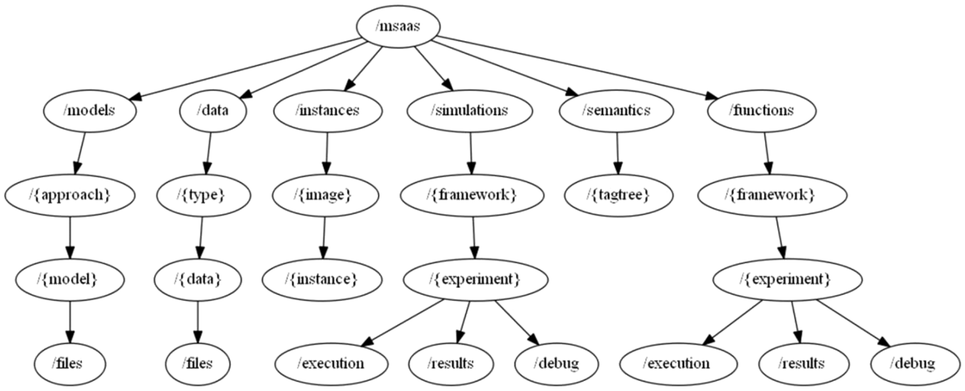


Fig. 8. The CloudRISE URI hierarchy. It contains six branches for the M&S resources (models, data, instances, simulations, semantics and functions).

There are six main branches in this hierarchy: *models, data, semantics, instances, simulations,* and*functions*. Each branch has its own hierarchy: *models* manages all the models in the Storage Units; *data* manages the data in the Storage Units; *semantics* manages a tag tree ontology to help composing resources; *instances* manages all available Compute Units; *simulations* provides environments of simulations, and *functions* provides environments of functions. In addition, these URIs of CloudRISE apply URI templates to deploy hierarchical resources. A URI template[56] uses variables placed between braces "{}". For instance, *approach* and *model* in the template *.../models/{approach}/{model}* can be substituted with any string (i.e., *.../models/devs/queue*). Please note that the branches in CloudRISE are not the same as the ones in the Storage Units (e.g., the AWS S3 in Fig. 7). The Cloud storage is only for static M&S resources in the Cloud, while the branches presented here use the services provided by the Cloud Layer and is able to expose resources as MSaaS services, allowing the users not to interact directly with the Cloud Layer. Unlike the Storage Units, the branches here also support the dynamic environments to reproduce experiments of simulations and functions.

CloudRISE works as repository interfaces for different Storage Units. For instance, it supports the repository of *models* for sharing all the models and *data* for the supported data. For each branch, CloudRISE maintains a URI template. For instance, */models* contains the list of modeling approaches. . . . */models/{approach}* specifies a particular modeling approach (e.g., DEVS, Modelica, Cell-DEVS), and *.../models/{approach}/{model}* specifies any model that belongs to a given approach. Users can use HTTP methods to operate the model URIs. For instance, PUT to provide an XML description for the model; POST to *.../models/{approach}/{model}/files* to append model files; DELETE to remove a model, or GET to retrieve it from the Cloud. These functions use the IaaS APIs provided by *AWSS3Storage.java* to connect with the corresponding Cloud storage. Similarly, for other supported data (e.g., system behavior files, scenario data and documentation), users can use *data* to share them (specifying the data type in *...data/{type}* and the data file in *.../data/{type}/{data}*).

In order to allow users to know better about the M&S resources, CloudRISE supports description files in XML. Let us explain the *Model Description* (an XML description for models) in detail. At present, CloudRISE implements DEVS atomic or coupled models. Both of them expose input ports and output ports for linking with other models. The structure of the Model Description looks as in Fig. 9. It keeps the basic information in *<Model>*, which includes model *name*, *type* (e.g., atomic or coupled), a *description* (a short paragraph to describe this model), and its *location* (e.g., the S3 URL). If the model contains input ports and output ports, it also keeps these *<Ports>* (*<Inputs> or <Outputs>*), including their *name*, *type*, and a *description*. In addition, it can store the files used for this model in *<Files>*, including file *name*, *type* and *location* (the URI in S3). A model can have multiple files; for instance, a DEVS model can have CPP classes, headers, and model

```
<ModelDescription>
      <Model name="" type="" description="" location=""/>
      <Inputs>
            <Portname="" type="" description="">
            …
      </Inputs>
      <Outputs>
            <Portname="" type="" description="">
            …
      </Outputs>
      <Files>
            <File name="" type="" location=""/>
            …
      </Files>
</ModelDescription>
```

Fig. 9. Structure of the Model Description file: basic information, input and output ports, and related files.

configuration files. The XML description shown here keeps general information, which can be customized if specific information is available (e.g., sub-models, coupling information).

The URIs following *.../msaas/instances* interface the MS-Images and instances of the Compute Units. As discussed earlier, an MS-Image is a copy of a Compute Unit consisting of CloudRISE, the simulators and the environments for supported functions. The URI *.../instances* is used to show the MS-Image available in the Compute Units; *.../instances/{image}* is used to show the instances that are launched from a specific image; and *.../instances/{image}/{instance}* is used to scale up and down the Cloud Computing units. Users can PUT to this URI to launch a new instance with the information mentioned in an XML description file (*Instance Description*); use GET to get the XML file; and DELETE to release an existing instance. The structure of the *Instance Description* file is as follows (see Fig. 10).

The file includes the instance *name*, *ID* (the EC2 instance ID in the AWS), *Image* (where it is launched from); *InstanceType* (computing power, different CPU,

```
    <Instance>
        <Name>MSaaS Instance 1</Name>
        <ID>i-b663c29c</ID>
        <Image>MS-Image 1</Image>
        <InstanceType>t1.micro</InstanceType>
        <IP>54.145.233.104</IP>
        <Status>running</Statue>
  </Instance>
```

Fig. 10. An example of Instance Description file: instance name, ID, image, instance type and status.

memory, storage, networking); *IP* (the IP of the image); and *Status* (e.g., running, stopped, terminated, initializing). This description file can be customized by users.

CloudRISE can improve the reproducibility of experiments by keeping an *EF Template* for each kind of experiment (i.e., the ***EF Template for Simulation*** for a simulation model, the ***EF Template for Function*** for a supported function). Each EF Template maintains a URI structure with EF information for users to reuse and reproduce experiments. The related EF information is specified in a corresponding XML file. These files are stored in the Storage Units, allowing the users to configure experiments for the simulation models and supported functions dynamically, and then control their life cycles.

For experiments with simulation models, *.../msaas/simulations* uses an URI structure called EF Template for Simulation (Fig. 11), which provides a hierarchical structure for the resources involved in an experiment of a simulation model.

This template contains the EF information and the experiments corresponding to it. The URI *.../msaas/simulations* is used to list all the models that have simulation experiments. The URI *.../msaas/simulations/{framework}* provides the EF Description, while *.../msaas/simulations/{framework}/{experiment}* is used to specify actual parameter values for an experiment. The URI *.../msaas/simulations/{framework}/{experiment}/execution* is used to reproduce the experiment, *.../{experiment}/results* to retrieve experiment outputs and *.../{experiment}/debug* for debug information.

The EF Template for Simulation, presented in Fig. 12, includes an EF Description for Simulation, which is an XML file to represent the context under which a system or a model is observed or experimented with Ref. 16. The EF can help to describe and reproduce the experiments. As shown in the example (Fig. 12), it

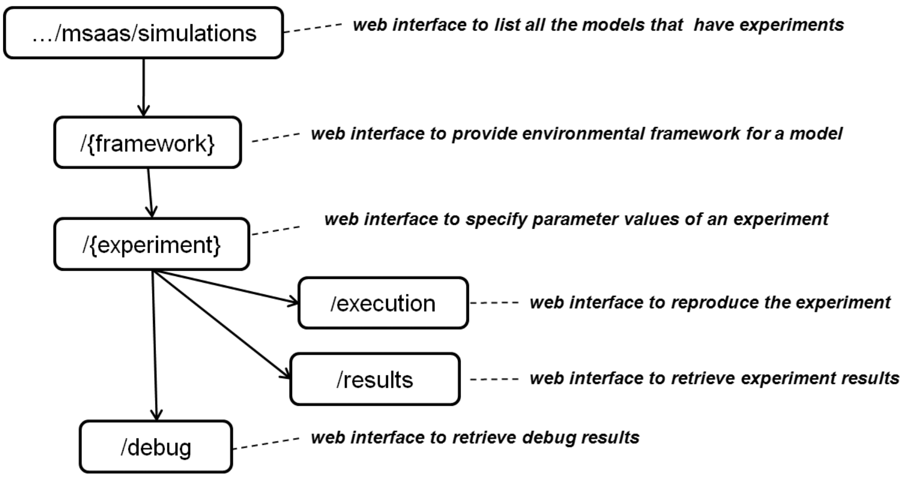

…/msaas/simulations
web interface to list all the models that have experiments
/{framework}
web interface to provide environmental framework for a model
/{experiment}
web interface to specify parameter values of an experiment
/execution
web interface to reproduce the experiment
/results
web interface to retrieve experiment results
/debug
web interface to retrieve debug results

Fig. 11. URI hierarchy of the EF Template for Simulation. RESTful WS interfaces to set up the framework, specify parameters of each experiment, execute experiment and get simulation results.

```
<Framework>
   <Name>…</Name>
   <Model>…</Model>
   <Context>
       <Description>…</Description>
       <Objective>…</Objective>
       <Assumption>…</Assumption>
       <Constrain>…</Constrain>
   </Context>
   <Inputs>
     <Parameter name=“” type=“” description=“”/>
     …
   </Inputs>
   <Outputs>
     <Parameter name=“”type=“” description=“”/>
     …
   </Outputs>
   <TerminateTime>…</TerminateTime>
   <Environments>
    <Environment group=“” instance=“” simulator=“” IP=“” port=“”/>
    …
  </Environments>
</Framework>
```

Fig. 12. Structure of the EF Description for Simulation: framework information, model context, input and output parameters definition, and environments information.

contains the following information: (1) The framework <*Name*>. (2) The <*Model*> to be simulated (its URI in the CloudRISE). (3) The <*Context*> of the model, including a <*Description*> its <*Objective*> any <*Assumption*> made by the designer, and <*Constrains*> under which the model was designed to operate. (4) The <*Input*> and <*Output*> parameters for the experiment. The input and output parameters in <*Parameter*> indicate which information it can receive and produce. Each parameter includes a parameter name, type and a short description. (5) The <*TerminateTime*>. (6) The simulation <*Environment*>: an *instance* to specify the Compute Unit (e.g., an EC2 instance), a *simulator* on that Compute Unit (e.g., CD++), the *IP* and *Port* CloudRISE running on the Compute Unit (e.g., http://www.amazon-ec2ip.com:8080), and a *group* to combine the Compute Units for running experiments.

A group can have different Compute Units. If a group has only one Compute Unit, then during the run-time, the Compute Unit runs complete experiments using the assigned simulator. If a group has more than one Compute Units, then these Compute Units within a same group run distributed simulation for the model. In that case, the model is partitioned into different sub-models; each Compute Unit executes one sub-model. CloudRISE reuses the simulation protocols in RISE for the distributed simulation, which can transmit messages and synchronizes the simulation time among Compute Units.[39] Users can extend the EF Description for Simulation for specific purposes.

Each simulation model can have different EFs with different contexts; and each EF can have different experiments with different parameter values according to runtime simulation scenarios. The *.../{framework}/{experiment}* is the URI for specifying actual parameter values of an experiment by using the *Experiment Description for Simulation*. Users can use PUT to this URI to create the Experiment Description for Simulation; POST to append needed inputs files; GET to retrieve this description file. As shown in the example below, the Experiment Description for Simulation contains the following information (see Fig. 13): the experiment <*Name*>, a short <*Description*>, the corresponding EF Description for Simulation in <EF>, the group number of the environment to run this experiment, input/output parameter values in <*Parameter*>, and number of runs in <*NumOfRuns*>. The values of input/output parameters can be the URIs of the Storage Units. Users can also upload files as input parameter values at runtime. Each Compute Unit assigned in the corresponding EF file runs the experiments in threads (with the number of <*NumOfRuns*>).

The EF Template for Simulation allows users to reproduce the experiments and control their lifecycles (Fig. 14). The user can provide the EF Description for Simulation to its *.../msaas/simulations/{framework}*, and specify its actual value for its parameters in *...simulations/{framework}/{experiment}*. Initially, the state of this experiment is *Init*. Users can check the state anytime by using GET from the URI *.../{experiment}/execution*. CloudRISE will copy all the existing files specified in the Experiment Description for Simulation to the workspace of the experiment; besides, users can also POST inputs for the experiment. After all of the required inputs are properly handled (from *.../data/* or *POSTed* by the user), the state is *Ready*. Then, users can use PUT to *.../{experiment}/execution*, and CloudRISE will start the simulation and change to *Running*. During the execution,

```
<Experiment>
      <Name>…</Name>
      <Description>…</Description>
      <EF>…</EF>
      <Environment group="“"/>
      <NumOfRuns>…</NumOfRuns>
      <Inputs>
            <Parameter name="“" value="“"/>
            …
      </Inputs>
      <Outputs>
            <Parameter name="“" value="“"/>
            …
      </Outputs>
</Experiment>
```

Fig. 13. Structure of the Experiment Description for Simulation: EF, run numbers, environment and input/output parameter values for the experiment.

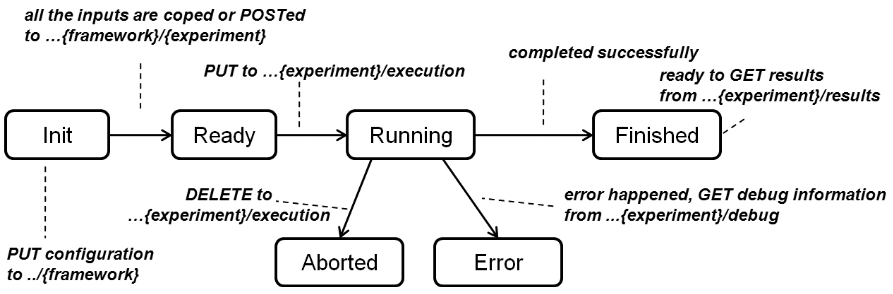

all the inputs are coped or POSTed to …{framework}/{experiment}
PUT to …{experiment}/execution
completed successfully
ready to GET results from …{experiment}/results
Init
Ready
Running
Finished
DELETE to …{experiment}/execution
error happened, GET debug information from …{experiment}/debug
PUT configuration to ../{framework}
Aborted
Error

Fig. 14. Lifecycle of the experiment in CloudRISE. Users control a specific experiment among different lifecycle phases (including Init, Ready, Running, Finished, Aborted, Error).

users can use DELETE to *…/{framework}/execution* to abort the simulation. After the experiment completes successfully, it turns to *Finished* and the user can retrieve output from *…/{experiment}/results*. If an error happens during its executing, it changes to the state *Error*.

Using these MSaaS services users can access and control its lifecycle easily, which can improve the reproducibility of experiments. If the model and data already exist, and the simulators are already configured, users only need to provide the EF Description and the Experiment Description for Simulation (for specifying actual parameters values of an experiment). Then, they can invoke the MSaaS services to reproduce the experiment and control its lifecycle as discussed above.

The experiments with the supported functions are shared and managed by *functions*. The EF Template for Function is similar to the EF Template for Simulation presented in Fig. 11. For example, *…/msaas/functions* is used to list all the functions that have experiments. The URI *…/msaas/functions/{framework}* provides the EF Description for a function. The URI *…/function/{framework}/{experiment}* is used to provide the Experiment Description for Function, which specifies actual parameter values of an experiment following its EF. The lifecycle of the supported function is also similar to the ones for the simulations presented in Fig. 14. There are two differences. The first one is that instead of the predefined simulators in the MS-image, users can POST their function programs at runtime by POSTing to *…/msaas/functions/{framework}*. Another difference is that in the EF Description for Function, the "group" in <*Environment*> has only one Compute Unit, which means that each function is run completely in one Compute Unit; in addition, the <*environment*> contains a new <*Command*> to specify about how the function executes in the environments. In the current implementation, it is the command line of the function. For example, in results parser function, a command line specified in <*Command*> like*…/{Function}* *{SimulationResults}* *{ParsedSimulationResults}* means that during the runtime, if the experiments of the function start, then it does the following:

(1) replace the variables with the parameters values specified in the Experiment Description for Function;

Table 4. Comparison among DEVS/SOA, RISE and CloudRISE.

	DEVS/SOA	RISE	CloudRISE
Domain	WBS	WBS	CBS
Supported Simulator	DEVS-based	DEVS-based, open to non-DEVS	DEVS-based, open to non-DEVS
Web framework	SOAP-based	RESTful WS	RESTful WS
VM	DEVS VM kernel	No	MS-Image
M&S resources	DEVS distributed simulation	DEVS distributed simulation	Varied M&S resources (MSaaS)
Experimental-oriented framework	Partial	Partial	Full

(2) call the command line in the assigned Compute Units to execute the experiments of the function.

As discussed in Sec. 2, many efforts in WBS expose simulation functions as WS, but at this time, CloudRISE is the first one to deploy DEVS simulation environment and other M&S resources as services in the Cloud, as discussed in Table 4.

The differences are as follows:

(1) DEVS/SOA and RISE are original designed for WBS, providing DEVS-based simulation protocol and environment through WS; while CloudRISE is designed for CBS, deploying different M&S resources as MSaaS in the Cloud.
(2) All of them currently provide DEVS simulation, but RISE and CloudRISE also have the potential to non-DEVS simulators. As discussed in Ref. 57, in order to add a new simulator, we can upload this simulator to an MS-Image, add it to a new branch .../{*simulator*} hierarchy in RISE/CloudRISE; then this MS-Image is ready to be used for the new simulator.
(3) DEVS/SOA provides simulation services as SOA-based WS; while RISE and CloudRISE provide services as RESTful WS. As discussed before, compared to RESTful WS, SOAP WS have the issues of structure constrains.
(4) As discussed in Table 3, DEVS/SOA uses DEVS VM kernel to provide a general DEVS simulation environment; while CloudRISE uses MS-Image. In contrast, RISE does not have such VM feature.
(5) CloudRISE also support users to deploy varied M&S resources as services, like model, data, functions, images, instances, and experiment frameworks.
(6) DEVS/SOA and RISE can use XML to configure experimental frameworks partially, while CloudRISE supports full control of experimental frameworks using XML configuration, such as models context, framework/experiment template and lifecycle control of experiments.

6. Case Study: Crowd M&S Using CloudRISE in the Cloud

In this section, we present a complete case study of Crowd M&S using the CAM-SaaS architecture, based on the CloudRISE middleware. We show how to use

CloudRISE in the Cloud to reuse varied resources and reproduce experiments involved in the Crowd M&S. We will discuss the implications using a case study representing a crowd model in a shopping mall with multiple floors.[58,59] In this model, the crowd tends to move forward, but they can change direction if the way blocked.[60]

Crowd M&S has been used to support the analysis of the behavior of crowds. It can be used to predict the impact of pedestrian movement and to test design alternatives.[60] Designers can use Crowd M&S to test their designs and find flaws before the construction has begun. To do this, different kinds of M&S resources are involved (Fig. 15).

To study the behavior of a crowd in the design of a building, we need basic information of the building (for instance, the floor plan of a building could be used as an input for a crowd model simulation). At present, new Building Information Modeling (BIM) tools are widely used to manage the buildings' data and improve the quality of building designs.[61] The Industry Foundation Classes (IFC) is a standard used to represent BIM files. We can extract information from IFC files, and can use them as inputs for crowd models. In addition, BIM tools can also be used for 3D visualization, and, in our case, for visualizing crowd simulation results.[62] By doing this, the designers can efficiently view, analyze and refine the design with different scenarios.

In general, the Crowd M&S process involves the following resources:

(1) The ***CrowdModel*** that describes the crowd behavior. In our case, we will use a Cell-DEVS model.
(2) The supported data of ***IFC*** file that contains the building information.
(3) The supported data of Domain Specific Models (***DSM***) file that contains the particular information for crowd simulation purposes.
(4) The supported function of ***BIM Data Collection***. It can extract data from the IFC designs and DSM files. The extracted data can be used as the crowd model input.

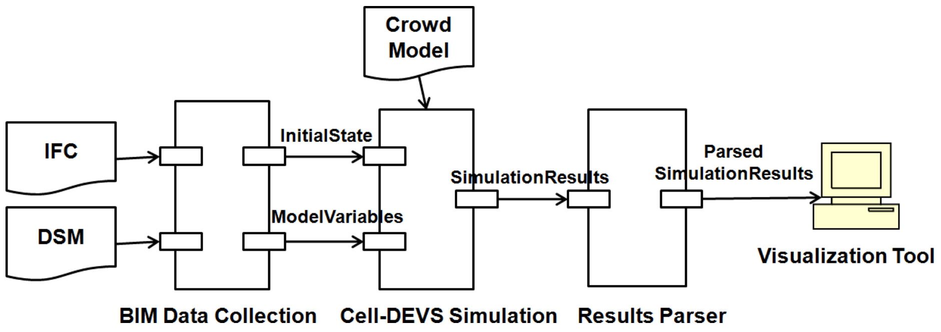


Fig. 15. The resources involved in the Crowd M&S: crowd model, IFC and DSM files (supported data), data collection function, Cell-DEVS simulation, simulation result parser.

(5) The experiments of BIM Data Collection. They can generate two outputs: the ***InitialState*** file that contains the layout of the building for the crowd model; and the ***ModelVariables*** that contains important values of the execution of the crowd model.
(6) The ***Simulation*** (in our case, a Cell-DEVS simulation). We can execute crowd simulations for the given model, with the inputs of *InitialState* and *ModelVariables.*
(7) The experiments of the crowd Simulation. They generate the ***SimulationResults***.
(8) The supported function called ***ResultsParser,*** which might need to be used parse the simulation results so they can be visualized in a 3D Visualization Tool.

In our case, all these resources are stored in Storage Units in the CloudRISE, including the resources provided by users (e.g., the crowd model), and the resources generated during the experiments (e.g., the simulation results). CloudRISE works as repository interfaces to expose these resources as MSaaS services. Following, we will briefly discuss how to use CloudRISE to execute the steps above.

In Refs. 58 and 59, we developed a crowd model in Cell-DEVS to study the crowd behavior during emergency in a multi-floor building. This model has walls, furniture, exits (on the first floor) and stairs (connecting floors). In the case of emergencies, people try to evacuate each floor, moving down through the stairs, and finally leave the building through the exits. The Cell-DEVS model was implemented in CD++ (with file name *crowdModel.ma*).[55] The behavior of each cell depends on its current state, determined by a set of rules after satisfying a precondition of his neighborhood. The rules of this model can be categorized as follows: (a) Someone enters a cell; (b) Someone leaves a cell; (c) Someone moves from a cell to a stairwell; (d) Someone enters a stairwell; and (e) Someone exits a stairwell.

We used CloudRISE, and activated the API of *.../models* to share the model file. The developer of the crowd model can POST to *.../models/cell-devs/crowd/files* while others can GET it from *.../models/cell-devs/crowd.* In addition, the developer of the model can provide an XML description file for this model via PUT to *.../models/cell-devs/crowd.* The Model Description allows users to know better about this model. They can obtain this description via GET to *.../models/cell-devs/crowd.* The Model Description for this crowd model is shown in Fig. 16.

As we can see, it contains the model name, its type and description, and its location in the Storage Unit. Note that in <*Model*>, the *location* is the model's folder, which contains the files related to this model (specified in <*Files*>). In our case, we have a file called *crowdModel.ma.* Note that the description file shown here keeps general information, which can be customized if specific information is available.

```
<ModelDescription>
       <Model name="CrowdModel" type="cell-devs" description="a cell-devs model to
       simulate crowd behavior in a multi-floor building" location=
       "…s3/MSaaS/Models/Cell-DEVS/crowd"/>
       <Files>
              <File name="CrowdModelDefinition" type="MAfile" location="…/crowdMode
              l.ma"/>
       </Files>
</ModelDescription>
```

Fig. 16. Model Description file for the crowd model. It specifies the model and related file information.

For other resources that are provided by users (e.g., the IFC file, the supported data of DSM, etc.), users can share them by using CloudRISE in a similar way. For instance, we can use the IFC files to manage the layout information of the area under study. Figure 17 shows the IFC file that we are interested to describe using Autodesk Revit (a BIM tool that can manage IFC files). Similarly, we can share this IFC file using the CloudRISE API *…data/ifc/buildingMall/files*. For the resources generated during the experiments at runtime, CloudRISE uses the parameter value of the Experiment Description to obtain/save them in the Storage Units.

As discussed in the previous section, CloudRISE can improve the reproducibility of experiments by keeping the EF Templates of experiments (for either simulations or supported functions). Following, we show the EF Templates for crowd simulations

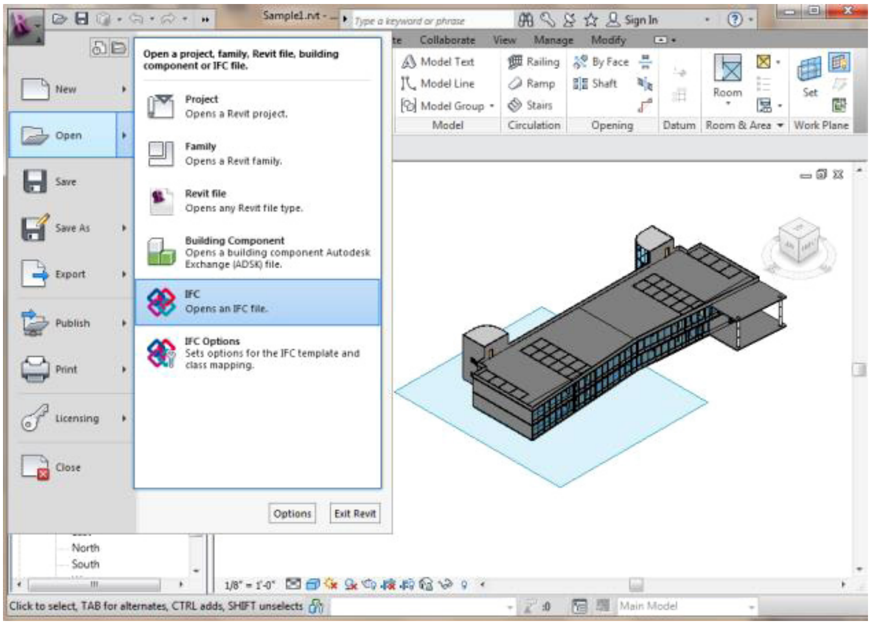


Fig. 17. IFC loaded in Autodesk Revit: crowd behavior during emergency in a shopping mall.

```
<Framework>
   <Name>EF of the crowd model</Name>
   <Model>…msaas/model/cell-devs/crowd<Model>
   <Context>
       <Description>a model to simulate crowd …</Description>
       <Objective>to study crowd behavior during …</Objective>
       <Assumption>the crowds move at a constant speed…</Assumption>
       <Constrain>cell sides should less than500x500x10</Constrain>
   </Context>
   <Inputs>
       <Parameter name="InitialState" type="VAL file" description="the initial
state of cells for the crowd model"/>
       <Parameter name="ModelVariables" type="MACRO file" description="dimension
variables for the crowd model" />
   </Inputs>
   <Outputs>
       <Parameter name="SimulationResults" type="LOG file" description="log of the
crowd simulation results"/>
   </Outputs>
   <TerminateTime>30 mins</TerminateTime>
   <Environments>
       <Environment group="1" simulator="CD++" instance="MSaaS Instance
1"IP="54.167.165.82" port="8080"/>
       <Environment group="2" simulator="CD++3.0" instance="MSaaS Instance 2"
IP="54.167.144.86" port="8080"/>
   </Environments>
</Framework>
```

Fig. 18. EF Description for the Crowd Simulations: crowd model location, model context, input and output parameter definition, and environments information.

(i.e., the *Cell-DEVS Simulation* component in Fig. 15). CloudRISE uses the URI *.../msaas/simulations/crowdEF* to maintain an EF Template for crowd simulations. This template contains an URI structure with EF information for users to reuse and reproduce experiments. The URI *.../msaas/simulations/crowdEF* provides the EF Description file shown in Fig. 18. Users can PUT and GET the EF Description via the URI of*.../msaas/simulations/crowdEF*.

This EF Description contains the following information: (1) The <*Name*>. (2) The <*Model*> to be simulated (its URI in the CloudRISE). (3) The <*Context*> of the model, including <*Description*>, <*Objective*>, <*Assumption*> and <*Constrains*>. (4) The <*Input*> parameters (i.e., *InitialState* and *ModelVariables*) and <*Output*> parameters (i.e., *SimulationResults*) for the experiment. Each parameter includes a name, type and a short description. These parameters indicate that the crowd simulation can receive the initial states and the dimension variables as the inputs of the crowd model, and then it can produce simulation results. (5) The <*TerminateTime*>. (6) the simulation <*Environments*> with two groups of EC2 instances for reproducing the experiments: one group has an instance with the CD++ simulator; the other has an instance with the CD++3.0 simulator.

After that, we can use other URIs in the EF Template for reproducing experiments. In particular, *.../msaas/simulations/crowdEF/evacuation* is used to

```
<Experiment>
      <Name>Evacuation experiment of a crowd model</name>
      <Description>an evacuation experiment of Crowd M&S in a three floors build-
ing </Description>
      <EF>…/simulation/crowd/evacuation/</EF>
      <Environment group="1"/>
      <NumofRuns>5</NumOfRuns>
      <Inputs>
            <Parameter              name="InitialState"              value="…s3
      /experiments/…/initialState1.val"/>
            <Parameter                name="ModelVariables"                 val-
      ue="…s3/experiments/…/modelVariables.macro"/>
      </Inputs>
      <Outputs>
            <Parameter
      name="SimulationResults"value="…s3/experiments/…/SimulationResults"/>
      </Outputs>
</Experiment>
```

Fig. 19. Experiment Description for the Crowd Simulations: EF, environment and input/output parameter values for the crowd experiment.

specify actual parameter values in an Experiment Description for Simulation. The URI *.../evacuation/execution* is used to reproduce the experiment. The URI *.../evacuation/results* is used to retrieve experiment outputs, and *.../evacuation/debug* is to get debug information. The Experiment Description for Simulation is shown in Fig. 19.

The description contains the experiment <*Name*>, a short <*Description*>, the corresponding EF Template URI in <EF>, the group number of the environment to run this experiment (i.e., it uses CD++ in *MSaaS Instance* 1), input/output parameter values in <*Parameter*>, and number of runs in <*NumOfRuns*>. The value of an input parameter is the corresponding URI in a Storage Unit; while the value of an output parameter is the folder URI in a Storage Unit (during runtime, this folder will save the results of five runs of this experiment).

CloudRISE allows users to scale up the instances and reproduce the experiments. As discussed in the previous section, users can use *.../msaas/instances* to do this. The users can use PUT to *.../instances/MS-Image*1/{*instance*} to launch new instances with the image of *MS-Image*1, and DELETE to release an existing instance. Users do not need to worry about details of launching/releasing instances, since CloudRISE uses IaaS APIs to interact with the underlying Cloud infrastructures. After the instances are ready, the users can use the EF Template mentioned above to reproduce the experiments and control their lifecycles. They can modify the instances as < *Environment* > in the EF Description (Fig. 18), and PUT — the EF Description to *.../msaas/simulations/crowdEF*. After that, they can modify the actual values of the parameters in the Experiment Description (Fig. 19), and PUT it to *.../msaas/simulations/crowdEF/evacuation*. Users can also POST inputs during runtime. Then, the state of the experiments is *Ready*. Users can use PUT to *.../msaas/simulations/crowdEF/evacuation/execution*, and

CloudRISE will start the simulation and change to *Running*. After the experiment completes successfully, it turns to *Finished* and the user can retrieve output from *.../msaas/simulations/crowdEF/evacuation/output*.

We conducted some tests focused on studying the simplicity of CloudRISE in terms of its environment configuration. The idea is to show how users can execute simulations easily using different instances and simulators, so that they can make decisions on the instance type and simulator for their demands (e.g., cost, time) quickly. Let us assume that users need to run a number of different simulation experiments for crowd behavior analysis using the model mentioned. The different users trying to conduct such analysis do not know which instance type and simulator they should use (they want to find a right combination to save money). Using CloudRISE, the users can easily run the experiments under different instance types and simulators, and find a right combination to use. Figure 20 shows the average execution time of different simulation experiments launched by CloudRISE. In this test, the same crowd model was run in three types of instances and each instance used two different simulators (i.e., there are six different combinations of instance type and simulator). Each combination run 5 experiments and the average execution time of these experiments is presented in the figure. The results show that different combination has different average execution time. Users launched three types of instances from an MS-image: *Micro Instances* (low-cost with small amount of CPU resources), *General Purpose Instances* (a balance of compute, memory and network resources), and *Compute Optimized Instances* (higher CPU cores and memory, but more expensive). Each instance automatically set up the CloudRISE middleware with two different simulation environments (*CD++* and *CD++3.0*). Then, the users simply uploaded EF and Experiment Descriptions as mentioned above, and they run experiments on each environment.

As we can see from Fig. 20, different combinations of instance type and simulator result in different average execution time. For instance, for each instance type, the

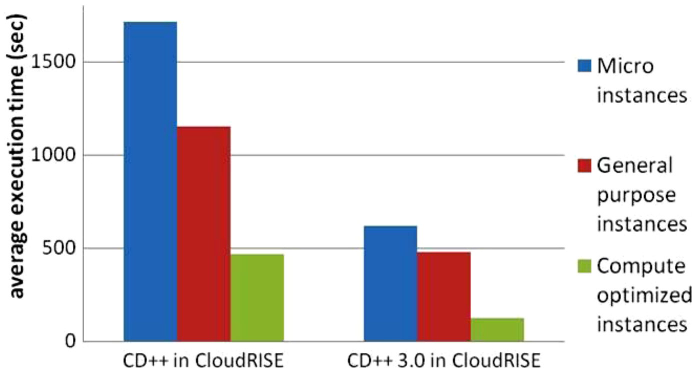


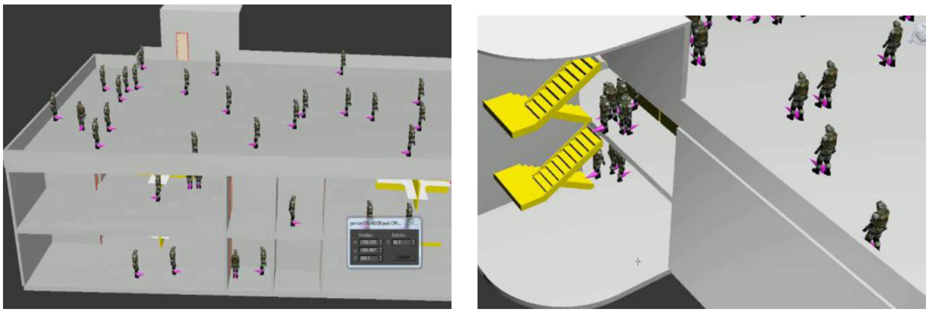
Fig. 20. Performance test of reproducing simulation experiments using CloudRISE. Users can quickly make decision on right instance type and simulator for their demands (e.g., cost, time).

CD++3.0 always cost less time than CD++ version (CD++3.0 has a faster way to collect simulation results and it does not use message queues as CD++ version does). In addition, for the experiments on either CD++ or CD++3.0, Compute Optimized Instances always have shortest execution time (i.e., 467s in CD++ and 124 in CD++3.0). These numbers are about 1/3 of the time required by General Purpose Instances and 1/5 of the time required by Micro Instances. It implies that if users highly care the execution time and they can afford some extra cost, they may choose Compute Optimized Instances; otherwise, if they want to save money and the performance of CD++3.0 in Micro Instances is acceptable, they may choose Micro Instances in the future.

The supported functions (in our case, *BIM Data Collection* and *Results Parser*) are shared and managed by *functions* in CloudRISE. The BIM Data Collection can extract information from *IFCBuildingFile* and *DSMInstance*, and generate *InitialState* and *ModelVariables* as the inputs of the crowd model. The EF Template for BIM Data Collection is in the URI of *…/function/datacollection/ifccollection*. The Results Parser can parse *SimulationResults* to *ParsedSimulationResults* (i.e., the format that can be used for 3D visualization). The EF Template for the Results Parser is in the URI *…/function/3dVisualization/parsingExperiment*. As mentioned in previous sections, the EF Templates for these functions are similar to the EF Template for the crowd simulation. The differences are that users can POST these function programs during the runtime, and they can provide a <*Command*> in the <*Environment*> of the EF Description. This *Command* can specify about how the function executes in the environment. For example, the command of BIM Data Collection is *java –jar {Function} -initLayout {InitialState} -initParameters {ModelVariables} -IFCFile {IFCBuildingFile} -DSMInstance {DSMInstance};* while the command of Results Parser is *./{Function} {SimulationResults} {ParsedSimulationResults}.* At runtime, when the experiment starts, CloudRISE will replace the variables with the actual values specified in the Experiment Description, and then execute the command line in the assigned instances.

At last, the parsed simulation results can be visualized in a 3D visualization tool. In Ref. 62, we developed a visualization tool in BIM (i.e., Autodesk 3ds Max). This tool can load the IFC file of the building and the parsed simulation results, and animate the crowd movements. Figure 21 shows how results look like: e.g., (a) crowd movements in different floors; and (b) crowd movements in a stairwell. The tool also allows the designer to filter specific floors and focus on individuals for better tractability and visibility. The demo of this case study can be found at *http://www.youtube.com/watch?v=u5idq-PDLck*.

In summary, a complete application using CloudRISE is available for studying the behavior of the crowds. CloudRISE can help the users to deploy different resources involved in the Crowd M&S as MSaaS services (e.g., the crowd model and the IFC file), and reproduce experiments (e.g., the crowd simulation, the data collection and the results parsing). More importantly, CloudRISE provides



(a) Crowds in different floors (b) Crowd moving in stairs

Fig. 21. 3D visualization of parsed results of crowd simulation. (a) Crowd movements in different floors; (b) crowd movements in a stairwell.

a scalable way to launch the instances to be used as experiment environments in the Cloud. Users can reuse these services to test different designs of buildings and determine which one is safest. For instance, they can provide different IFC files as services, configure different Compute Units in the Cloud, reproduce the experiments, and observe the parsed results in the 3D visualization tool.

7. Conclusion

We introduced a novel architecture, named the CAMSaaS. The CAMSaaS architecture is defined with the goal of simplifying the deployment process of varied M&S resources as services in the Cloud by taking full advantages of the Web. We presented the concept and implementation of CAMSaaS: the Cloud Layer (which focuses on the simplicity in terms of environment configuration) and the MSaaS Layer (which deals with the structure constrains caused by the web service framework and the variety of resources). The Cloud Layer provides IaaS services with a scalable Cloud infrastructure. The MSaaS Layer is a middleware based on RESTful WS that is able to use IaaS provided by the Cloud Layer and to deliver varied MSaaS services in the Cloud. We also presented the design and implementation details of CloudRISE, which extends from RISE and implements the concept of the MSaaS middleware. We illustrated a complete case study of Crowd M&S as an application using the CAMSaaS and its CloudRISE.

The proposed architecture has the following advantages:

(1) It provides a **hierarchical resource services** for varied M&S resources. CloudRISE uses a resource-oriented design by using RESTful WS. These resources are decomposed into hierarchical URIs, and they exchange information through standard XML messages via HTTP methods (GET/PUT/POST/DELETE). These RESTful WS take full advantages of the Web. Users can deploy their resources dynamically (i.e., they can be created and destroyed at runtime).

(2) It supports **experimental-oriented framework** for reproducing experiments (for either simulations or supported functions). The EF helps users to document the conditions of models and related experiments (e.g., context, inputs/outputs, and environment). CloudRISE uses the EF Template to manage the EF information and the related experiments. It allows users to reproduce and control the lifecycle of experiments.

(3) It uses IaaS to provide a **scalable infrastructure** for M&S resources. The Cloud Layer develops a simplified IaaS APIs of underlying Cloud infrastructure, which facilitate supper layers to access the Cloud. CloudRISE uses these IaaS APIs to save M&S resources in Storage Units and reproduce experiments in Compute Units. In particular, the MS-Image in the Cloud Layer can scale up/down environments for experiments by launching/releasing Compute Units.

(4) It uses MSaaS to make **everything as a service**. CloudRISE is built on top of the Cloud Layer. It allows users to deploy any resource related to M&S as service in the Cloud. For instance, its models/data/simulations/functions branches support Model as a Service, Data as a Service, EF as a Service, and Function as a Service respectively.

The long-term goal of CAMSaaS is to support semantic mashup in the Cloud, which allows users to select from the deployed MSaaS and other Web APIs based on the right semantic and then easily build M&S applications. The integration of CAMSaaS and semantic mashup in future will advance CBS in terms of deployment, discovery and invocation of MSaaS.

Acknowledgment

This research has been partially funded by NSERC.

References

1. Papelis Y., Madhavan P., Modeling human behavior, in *Modeling and Simulation Fundamentals*: *Theoretical Underpinnings and Practical Domains*, John Wiley & Sons, NY, pp. 14–23, 2010.
2. D'Angelo G., Parallel and distributed simulation from many cores to the public Cloud, *Int. Conf. High Performance Computing and Simulation*, Istanbul, Turkey, 2011.
3. Byrne J., Heavey C., Byrne P. J., A review of web-based simulation and supporting tools, *Simul. Model. Practice Theory* **18**(3):253–276, 2010.
4. Jung W., Kim S. I., Kim H. S., Ontology modeling for REST Open APIs and web service mash-up method, 2013 *Int. Conf. Information Networking* (*ICOIN*), Phuket, Thailand, pp. 523–528, 2013.
5. Onggo S., Taylor S., Tulegenov A., The need for cloud-based simulation from the perspective of simulation practitioners, *Proc. Operational Research Society Simulation Workshop* 2014 (*SW*14), Worchestershire, UK, pp. 103–112, 2014.
6. Strassburger S., Schulze T., Fujimoto R., Future trends in distributed simulation and distributed virtual environments: Results of a peer study, *Proc.* 2008 *Winter Simulation Conf.*, Miami, FL, pp. 777–785, 2008.

7. Taylor S. J. E., Fishwick P. A., Fujimoto R., Page E. H., Uhrmacher A. M., Wainer G., Panel on grand challenges for modeling and simulation, *Proc.* 2012 *Winter Simulation Conf.*, Washington, DC, p. 232, 2012.
8. Taylor S. J. E., Khan A., Morse K., Tolk A., Yilmaz L., Zander J., Grand challenges on the theory of modeling and simulation, *Proc. Symp. Theory of Modeling and Simulation*, San Diego, CA, p. 34, 2013.
9. Johnson H. E., Tolk A., Evaluating the applicability of cloud computing enterprises in support of the next generation of modeling and simulation architectures, *Proc. Military Modeling and Simulation Symp.* (*MMS* '13), San Diego, CA, p. 4, 2013.
10. Cayirci E., Configuration schemes for modeling and simulation as a service federation, *Simulation* **89**(11):1388–1399, 2013.
11. Cayirci E., Modeling and simulation as a Cloud service: A survey, *Proc.* 2013 *Winter Simulation Conf.*, Savannah, GA, pp. 777–785, 2013.
12. Liu X., He Q., Qiu X., Chen B., Huang K., Cloud-based computer simulation: Towards planting existing simulation software into the cloud, *Simul. Modell. Practice Theory* **26**(1):135–150, 2012.
13. Tolk A., Mittal S., A necessary paradigm change to enable composable cloud-based M&S services, 2014 *Winter Simulation Conf.*, Savannah, GA, pp. 356–366, 2014.
14. Wagh K., Thool R. A., comparative study of soap vs. rest web services provisioning techniques for mobile host, *J. Inform. Eng. Appl.* **2**(5):12–16, 2012.
15. Cheng Q., Huang J., Research on Cloud-based simulation resource management. *Proc.* 2013 *Chinese Intelligent Automation Conf.*, Yangzhou, China, pp. 569–576, 2013.
16. Zeigler B. P., Praehofer H., Kim T. G., *Theory of Modeling and Simulation*: *Integrating Discrete Event and Continuous Complex Dynamic Systems*, Academic Press, NY, 2000.
17. Skoogh A., Perera T., Johansson B., Input data management in simulation — Industrial practices and future trends, *Simul. Model. Practice Theory* **29**:181–192, 2012.
18. Guo S., Bai F., Hu X., Simulation software as a service and service-oriented simulation experiment, 2011 *IEEE Int. Conf. Information Reuse and Integration* (*IRI*), Las Vegas, NV, pp. 113–116, 2011.
19. Fielding R. T., Architectural styles and the design of network-based software architectures, Doctoral dissertation, University of California, Oakland, CA, 2000.
20. Song Y., Xu K., Liu K., Research on web instant messaging using REST web service, 2010 *IEEE* 2*nd Symp. Web Society* (*SWS*), Beijing, China, pp. 497–500, 2010.
21. Barros F. J., Lehmann A., Liggesmeyer P., Verbraeck A., Zeigler B. P., 04041 abstracts collection — Component-based modeling and simulation, *The Int. Conf. Research Center* (*IBFI*), Schloss Dagstuhl, Germany, p. 457, 2004.
22. Robinson S., Nance R. E., Paul R. J., Pidd M., Taylor S. J., Simulation model reuse: Definitions, benefits and obstacles, *Simul. Model. Practice Theory,* **12**(7):479–494, 2004.
23. Wang W., Wang W., Zhu Y., Li Q., Service-oriented simulation framework: An overview and unifying methodology, *Simulation* **87**(3):221–252, 2010
24. Fortmann-Roe, S., Insight marker: A general-purpose tool for web-based modeling & simulation, *Simulation Modelling Practice and Theory* **47**:28–45, 2014.
25. Smit M., Stroulia E., Simulating service-oriented systems: A survey and the services-aware simulation framework, **6**(4):443–456, 2013.
26. Bencomo S. D., Control learning: Present and future, *Ann. Rev. Control* **28**(1):115–136, 2004.
27. Fortmann-Roe S., Insight Maker: A general-purpose tool for web-based modeling and simulation, *Simul. Model. Practice Theory* **47**:28–45, 2014.

28. Kuljis J., Paul R. J., Web-based discrete event simulation models: Current states and possible futures, *Simul. Gaming* **34**(1):39–53, 2003.
29. Papazoglou M., *Web Services*: *Principles and Technology*, Pearson Prentice Hall, 2007.
30. Richardson L., Ruby S., *RESTful Web Services*, O'Reilly Media, USA, 2007.
31. Tsai W. T., Fan C., Chen Y., Paul R., DDSOS: A dynamic distributed service-oriented simulation framework1, *Proc. 39th Annual Symp. Simulation*, Washington, DC, pp. 160–167, 2006.
32. Brebner P., Service-oriented performance modeling the MULE enterprise service bus (ESB) loan broker application, *Proc. 35th Euromicro Conf. Software Engineering and Advanced Applications*, Patras, Greece, pp. 404–411, 2009.
33. Madhoun R., Web-services definition of discrete-event simulation services, Master thesis from Systems and Computer Engineering, Carleton University, 2006.
34. Madhoun R., Feng B., Wainer G., On the creation of distributed simulation web-service-based distributed CD++, *Proc. Artificial Intelligence*, *Simulation and Planning*, Buenos Aires, Argentina, pp. 1–6, 2007.
35. Wainer G., Madhoun R., Al-Zoubi K., Distributed simulation of DEVS and Cell-DEVS models in CD++ using web-services, *J. Simul. Model. Practice Theory* **16**(9):1266–1292, 2008.
36. Mittal S., Risco-Martín J. L., Zeigler B. P., DEVSML: Automating DEVS execution over SOA towards transparent simulators, *Proc. 2007 Spring Simulation Multiconference*, Norfork, VA, pp. 287–295, 2007.
37. Mittal S., Risco-Martin J. L., *Netcentric System of Systems Engineering with DEVS Unified Process*: *A Book in System of Systems Engineering*, CRC/Taylor & Francis, 2013.
38. Muqsith M. A., Sarjoughian H. S., A simulator for service-based software system co-design, *Proc.* 3*rd Int. ICST Conf. Simulation Tools and Techniques*, Malaga, Spain, p. 54, 2010.
39. Al-Zoubi K., Wainer G., RISE: A general simulation interoperability middleware container, *J. Parallel Distrib. Comput.* **73**(5):580–594, 2013.
40. Arroqui M., Mateos C., Machado C., Zunino A., RESTful web services improve the efficiency of data transfer of a whole-farm simulator accessed by android smartphones, *Comput. Electron. Agric.* **87**:14–18, 2012.
41. Al-Zoubi K., Wainer, G., Performing distributed simulation with RESTful web services. *Proc. 2009 Winter Simulation Conf.*, Austin, TX, pp. 1323–1334, 2009.
42. Wainer G. A., Al-Zoubi K., Dalle O., Mittal S., Martín J. L. R., Sarjoughian H., Zeigler B. P., Standardizing DEVS simulation middleware, in Wainer G., Mosterman P. (eds.) *Discrete-Event Modeling and Simulation*: *Theory and Applications*, Taylor and Francis, UK, P. 459, 2010.
43. Mulligan G., Gracanin D., A Comparison of SOAP and REST implementations of a service based interaction independence middleware framework, *Proc.* 2009 *Winter Simulation Conference* (*WSC*), Austin, TX, pp. 1423–1432, 2009.
44. Feng X., Shen J., Fan Y., REST: An alternative to RPC for Web services architecture, in *First Int. Conf. Future Information Networks*, Beijing, China, 2009.
45. Siriwardena P., *Advanced API Security*, Springer Press, NY, 2014.
46. Fujimoto R. M., Malik A. W., Park A. J., Parallel and distributed simulation in the cloud, *SCS MS Mag.* **3**:1–10, 2010.
47. Tsai W. T., Fan C., Chen Y., Paul R., DDSOS: A dynamic distributed service-orinted simulation framework 1, in *Proceedings of the 39th Annual Symposium on Simulation*, Washington, DC, pp. 160–167, 2006.

48. Sliman L., Charroux B., Stroppa Y., A new collaborative and cloud based simulation as a service platform: Towards a multidisciplinary research simulation, 15*th Int. Conf. Computer Modelling and Simulation*, Cambridge, United Kingdom, pp. 611–616, 2013.
49. Li B. *et al.*, Networked modeling simulation platform based on concept of cloud computing — cloud simulation platform, *J. Syst. Simul.* **21**(17):5292–5299, 2009.
50. Bruneliere H., Cabot J., Jouault F., Combining model-driven engineering and cloud computing, in *Modeling, Design, and Analysis for the Service Cloud — MDA4ServiceCloud'*10: *Workshop's* 4*th edition*, Paris, France, pp. 1–2, 2010.
51. LannerGroup, http://www.lanner.com/en/l-sim.cfm/, Accessed on Feb. 28, 2016.
52. CIMdata, http://www.cimdata.com/en, Accessed on Feb. 28, 2016.
53. Garg S. K., Versteeg S, Buyya R, SMICloud: A framework for comparing and ranking cloud services, *Fourth Int. Conf. Utility and Cloud Computing*, Melbourne, Australia, pp. 210–218, 2011.
54. Kurkowski S. H., Credible mobile *ad hoc* network simulation-based studies, Doctoral Dissertation, Colorado School of Mines, 2006.
55. Wainer G., *Discrete-event Modeling and Simulation: A Practitioner's Approach*, CRC/Taylor & Francis, UK, 2009.
56. Gregorio J., Fielding R. T., Hadley M., Nottingham M., Orchard D., RFC6570: URI template, Internet Engineering Task Force (IETF) Request for Comments, 2012.
57. Wang S., Wainer, G., A simulation as a service methodology with application for crowed modeling, simulation and visualization, *Simulation: Transactions of the Society for Modeling and Simulation* **91**(1):71–95, 2015.
58. Wang S., Schyndel M. V., Wainer G., Subashini V., Woodbury R., DEVS-based building information modeling and simulation for emergency evacuation, *Proc.* 2012 *Winter Simulation Conf.*, Berlin, Germany, pp. 1–12, 2012.
59. Wang S., Wainer G., Goldstein R., Khan A., Solutions for scalability in building information modeling and simulation-based design, *Symp. Simulation for Architecture and Urban Design*, San Diego, CA, p. 7, 2013.
60. Hoogendoorn S. P., Bovy, P. H. L., Pedestrian route-choice and activity scheduling theory and models, *Transport. Research Part B: Method.* **38**(2):169–190, 2004.
61. Ham N. H., Min K. M., Kim J. H., Lee Y. S., Kim J. J., A study on application of BIM to pre-design in construction project, 3*rd Int. Conf. Convergence and Hybrid Information Technology*, Busan, Korea, pp. 42–49, 2008.
62. Freire V., Wang S., Wainer G., Visualization in 3ds max for cell-DEVS models based on moving entities, *Symp. Simulation for Architecture and Urban Design*, San Diego, CA, p. 9, 2013.