Journal of Computational Science 21 (2017) 113–131

Contents lists available at ScienceDirect

Journal of Computational Science

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

MAMS: Mashup architecture with modeling and simulation as a service

Gabriel Wainer*, Sixuan Wang

Department of Systems and Computer Engineering, Carleton University, Ottawa, ON K1S-5B6, Canada

ARTICLE INFO

Article history:
Received 13 September 2016
Received in revised form 8 May 2017
Accepted 20 May 2017
Available online 29 May 2017

Keywords:
Modeling and simulation as a service
Cloud-based simulation
Mashup
Web data integration
Web service composition

ABSTRACT

The Modeling and Simulation (M&S) community has used Web service (WS) technologies to help with the increasing complexity of the applications. Although there are many Web APIs that could be useful for M&S applications, they are rarely used. One of the main reasons is that using M&S related WSs and open APIs makes it difficult to consider their integration for building complex M&S applications. Here we show how to simplify the development and integration process of such M&S applications by using a new architecture for web services Mashups, namely the Mashup Architecture for Modeling and Simulation (MAMS). MAMS is the first environment and architecture to integrate mashups and distributed simulation. MAMS provides means for a lightweight life cycle to develop, deploy, identify, select, integrate and execute varied M&S resources as services in the Cloud. We present different case studies showing the utility of the proposed architecture.

1. Introduction

Modeling and Simulation (M&S) has become a discipline with its own body of knowledge, formalisms and methodologies and nowadays M&S applications are becoming more and more popular and complex. This resulted in various issues.

1) Existence of **varied M&S resources**: The quantity and variety of the M&S resources available are increasing. There are now many source systems, models, simulators, and experimental frameworks, as well as other related data (e.g., text, database) and functions (e.g., data collection, result analysis tools) [19] available to build simulation software.
2) Complex **environment configuration**: we need to configure and maintain simulation environments to run varied model experiments under different scenarios [10].
3) An **increasing number of users**: more and more people are involved in M&S activities [5]. They can be specialists or end users who need such service.

In order to help with the development process for M&S applications, various mechanisms have been introduced, such as the M&S life cycle [5], and Model Engineering [49]. However, these mechanisms might result in complex M&S life cycles.

In order to simplify the development process of M&S applications, in the last 20 years, the M&S community started using distributed computing, developing Web-based Simulation (WBS), Cloud-based Simulation (CBS) and Web Service (WS) technologies. WBS focused on running simulation experiments using the Web and exposing M&S functions as web services [8]. This method has been successful, and a large number of M&S WSs exist. At the same time, the emergence of Cloud Computing made Cloud-based Simulation (CBS) an alternative to WBS [23]. CBS integrates WBS and Cloud Computing. CBS uses Cloud computing to manage varied M&S resources and build different simulation environments [9], making it easier to develop M&S applications by exposing M&S resources as **Modeling and Simulation as a Service** (**MSaaS**).

The Web community has developed many open Web APIs. One provider, ProgrammableWeb.com, has over 11,000 APIs registered. Many of these open services (e.g., weather forecast, geographical information, and data analytics applications) could be useful for M&S applications but they have not have been fully exploited yet. This growth of M&S related WSs and open APIs makes it difficult to consider their integration for building complex M&S applications. Domain experts usually are not trained to use the existing M&S related resources and services easily. Although they could improve

* Corresponding author.
E-mail addresses: gwainer@sce.carleton.ca (G. Wainer), swang@sce.carleton.ca (S. Wang).

http://dx.doi.org/10.1016/j.jocs.2017.05.022
1877-7503/

the user experience and make richer applications [13], until now, there has been no research showing how to integrate these available M&S services and useful Web APIs in a simplified way [33]. Our goal is thus to simplify the development and integration process of M&S applications that involve such heterogeneous data and services. This should be a lightweight process, allowing the easy reuse and integration of existing resources and services. People from different domains should be able to use the process for sharing and combining their services quickly. In addition, the process should allow users to easily link and execute the services.

Based on these ideas, we presented an architecture based on mashup technologies. Mashups allow integrating different content from existing sources in the Web to create value-added applications [4]. Our goal is to use M&S mashups to integrate different services and resources related to M&S. Our architecture, called MAMS (Mashup Architecture with Modeling and Simulation as a Service) is the first to integrate mashups, RESTful Web Services and distributed simulation.

The M&S mashup was introduced in [45]; here present a detailed version with a thorough discussion, details about implementation of the architecture and tools, and a variety of case studies developed using the architecture. We present the MAMS architecture in detail, introducing the M&S mashup method, focusing on three layers of the architecture. We use an implementation of the architecture for rapid development of Web-based applications as mashups. We present different case studies, in order to show the feasibility of the approach. We want to develop new applications as composition of existing web services and we will discuss how this can be achieved.

The rest of this paper is organized as follows. In Section 2, we review the related work. In Section 3, we discuss the MAMS architecture. In Section 4, we present the M&S mashup method in details. In Section 5, we discuss the tools for M&S mashup. In Section 6, we show different applications using MAMS for rapid M&S mashup development.

2. Related work

M&S applications involve varied resources. We can categorize these resources in two groups: *entities* (i.e., basic resources that are directly related to M&S) or *support resources* (i.e., resources that are helpful for M&S). In our case, for *entities,* we will use the theory of modeling and simulation presented in [48], which provides a conceptual framework for entities. The entities in the theory are the *source system*, the *models*, the *simulators*, the *experimental frameworks* and the *experiments*. The source system is an entity 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 to study its behavior over time. Simulation involves different experimental frameworks and associated experiments. On the other hand, the *support resources* include various *data* or *functions*. Support data is any resource that is not executable, in the form of text, file, picture, video, etc [31]. Support functions are executable resources, such as data collection components, results analysis programs, and visualization tools [28].

M&S applications can be developed in different ways and different M&S life cycle management methods have been proposed in the last 50 years. Recently, Radeski and Parr [25]. proposed a modeling simulation life cycle with different phases including development, use, maintenance, and reuse of the M&S applications. In [5], the authors proposed another M&S life cycle built as an iterative framework. In [49], the authors proposed the concept of Model Engineering, which is defined as a general engineering methodology that guarantees the credibility of the full life cycle of a model. However, a common issue found in the methods mentioned above, is that as more and more resources are involved, the development process becomes more complex. Each method is composed of different phases, which can take a long time to develop.

In order to simplify the development of M&S applications, the M&S community started using web technologies, which resulted in the invention of Web-Based Simulation (WBS, exposing M&S functions) and Cloud-Based Simulation (CBS, integrating WBS and Cloud Computing).

In WBS, the simulator and its environment are located remotely. Users can submit requests (and parameters) to the simulator through web servers. The experiments run remotely, and the results are returned to the user. WBS eases the sharing of the resources: the engines are available on the server (so we do not need to worry about setting up a simulation environment or by software dependency issues). WBS also improves data accessibility, interoperability and user experience (non-expert users can access these services easily), allowing them increase productivity [9].

Web services technologies can be broadly categorized into SOAP-based [24], in which the service expose an arbitrary set of operations, and RESTful [26], in which we manipulate XML representations of Web resources using a uniform set of "stateless" operations. SOAP has been widely used in M&S systems. For instance, in [20,41], we defined the first distributed DEVS (Discrete-Event System Specification) simulation environment over SOAP. SOPM [6] is a SOAP-based modeling framework to build performance models of SOA. In [29], the authors developed a tool using SOAP-based web services to support the creation of training simulations. DEVS/SOA [21] implements DEVS over a SOAP-based SOA framework, supporting a development and testing environment known as the DEVS Unified Process. However, in recent years, the research and publications about WBS dropped quickly [35]; the main reason is there was a mismatch between SOAP-based WS and the main characteristics of the Web. Most WBS frameworks are based on SOAP, but SOAP has issues of structural constrains, e.g., tightly coupled services, bigger payload, exposing internal implementation, limited scalability, security issues, etc. [38]. Any SOAP-based WS project can take years for defining the SOAP WS layer, deploying those services, and standardizing the interfaces. Likewise, 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.).

Instead, RESTful WS imitate the Web interoperability style, taking full advantage of the Web. REST represents resources as URIs; any user can request the resource by HTTP methods, and communicate with the resource via standard representations. Resources are manipulated using a fixed set of operations: POST, GET, PUT, and DELETE. However, using REST for WBS has started only recently. The RESTful Interoperability Simulation Environment (RISE) [2] was the first RESTful distributed simulator. Its main objective is to support interoperability of distributed and heterogeneous simulations regardless of the model formalisms, languages or simulation engines. Arroqui et al. [3] developed an agricultural information system by using both RESTful WS and SOAP-based WS. They found that RESTful WS required 24% less bandwidth for transferring data than SOAP-based WS do, which is similar to the results presented in [2].

On the other hand, Cloud-based Simulation (CBS) is becoming popular to use varied M&S resources [33]. CBS captures the intersection between M&S and Cloud Computing, which is good for building simulation environments in a pay-as-you-go manner [9]. We can use Cloud Computing to manage varied M&S resources and to build different environments on demand. Cloud Computing for WBS also uses advanced technologies such as load balancing, fault tolerance and advanced security [11]. CBS can help us to develop Modeling and Simulation as a Service (MSaaS). MSaaS uses cloud

services, hiding the infrastructure, platform and software details [37].

Recent research focused on how to use Cloud and MSaaS for developing M&S applications. This research field can be classified in three areas:

1) **Managing M&S resources**. For instance, [18] focused on deploying existing M&S software into the Cloud. Sliman et al. [32] proposed an MSaaS platform called RunMyCode, which allows scientists to share their code associated with their research publications in the Cloud.
2) **Supporting Parallel and Distributed Simulation (PADS)**. For instance, Li et al. [17] proposed a Grid-based platform (Cloud Simulation Platform), and summarized 12 key technologies for the development of such platform. Taylor et al. [34] introduced the CloudSME simulation platform to transplant legacy Grid-based simulation software in the Cloud.
3) **Building applications in the Cloud**. Bruneliere et al. [7] presented an approach to build applications by using model-driven engineering and Modeling as a Service (MaaS) in the Cloud. CIMdata (http://www.cimdata.com/en) proposed a simulation application using the Cloud in the area of commercial product design.

Although there have been various advances, the use of Cloud Computing and MSaaS is still in a preliminary stage [9], and little effort has been done in integrating different M&S services [33]. Likewise, there are open Web APIs that may be helpful for M&S; they could be used to improve the user's experience (e.g., Google Maps, YouTube, Geographical Information Systems – GIS –, etc.). There are more than 11,000 APIs registered by ProgrammableWeb.com, 73% of which are RESTful while and the remaining 27% are SOAP-based [30]. The RESTful services are more scalable, interoperable and simpler [22]. In REST, it is convenient to handle resources with URI and HTTP, so its utility for mashups is better than SOAP. Although WADL is used to describe REST web services, many companies describe their REST APIs on HTML pages, such as Mashape (http://www.mashape.com/explore). SOAP WSs are usually described in WSDL files. For example, WebServiceX (http://www.webservicex.net/ws) has over 70 SOAP WSs using WSDLs. Nevertheless, until now there has been no research showing how to integrate these web services for M&S.

Mashup technologies can be used to help simplifying the development of M&S application on the Web. The idea of a mashup is to integrate different services, using content from more than one source from the web to create new value-added applications. Mashups integrate heterogeneous data, application logic, and UI components (e.g., widgets). A large number of mashup techniques and tools have been developed in both industry and academia [12]. Many companies have developed their own commercial mashup tools, like IGoogle (http://www.igoogleportal.com), and Yahoo! Pipes (http://www.pipes.yahoo. com). They are based on the visual connection of components of heterogeneous data at the enterprise level, offering Do-It-Yourself guidance to meet user requirements. Many academic projects use End-User Programming (EUP), focusing on the composition and integration of web sources. For instance, Mashroom [46] uses relational models and provides operations like merge and filter over tables. In [1], the authors use native language programming in mashup components, to link different logic together.

One fundamental element in current mashup technologies is the *widget*, a processing unit for performing single purpose tasks such as fetching, parsing, formatting and visualizing data [36]. For instance, DERI Pipes [16] enables users to build widgets to process data from different sources (e.g., RDF, SPARQL, XML, HTML). In [36], the authors proposed an open mashup platform with linked widgets created by users that can be discovered and combined. WireCloud [47] is an open source mashup platform provided by the FI-WARE project that can implement widgets and build mashups by "wiring" widgets. For example, http://www.100widgets.com provides different widgets, supporting varied type of data (i.e., forms, diagrams, maps, photos and videos).

In [10], the authors compared traditional development and mashups for building web applications. A traditional process, shown in Fig. 1(a), uses various sub processes, (e.g., Requirements analysis, Design, Implementation, Testing and evaluation, Usage and maintenance), which is time-consuming. The mashup development process in Fig. 1(b) is simpler. It is prototype-centric and iterative: the user mashes up existing resources and runs the result; in case of unsatisfactory results, the user fixes the problems and runs the mashup again. The authors also showed that a key for mashup development is to have a platform to select and compose existing resources, along with a set of open Web APIs that support features and available data.

Mashup technologies can be useful for integrating available services (e.g., WBS, CBS and Web APIs) for M&S applications. Unfortunately, existing mashup techniques and tools cannot work directly for M&S because they are domain specific (the widgets do not support MSaaS and different kinds of Web APIs). In this research, *we propose a new mashup method focused on the process*

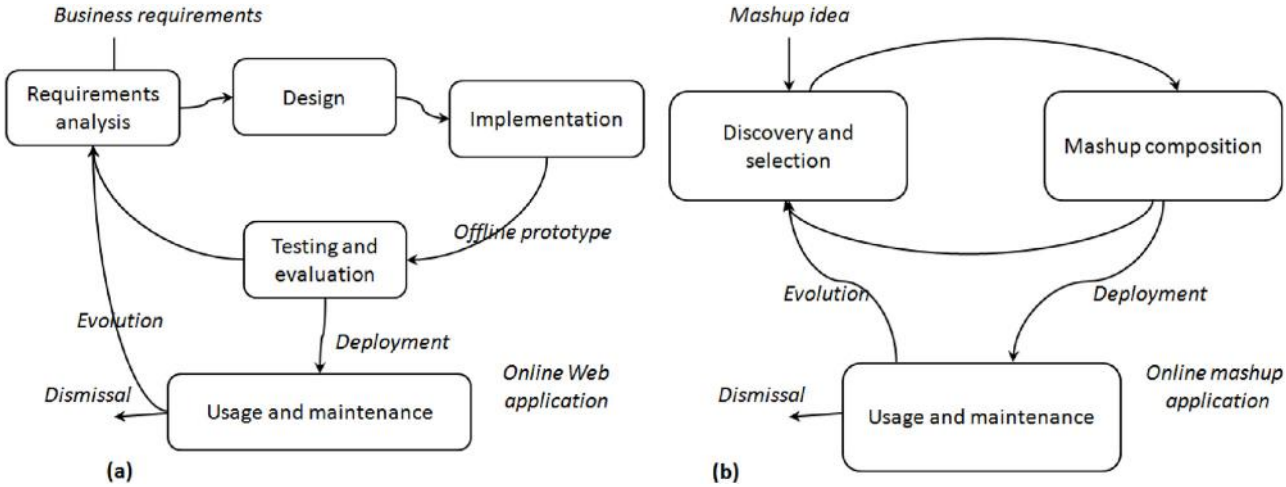


Fig. 1. Life cycles of (a) traditional web applications and (b) mashups [16].

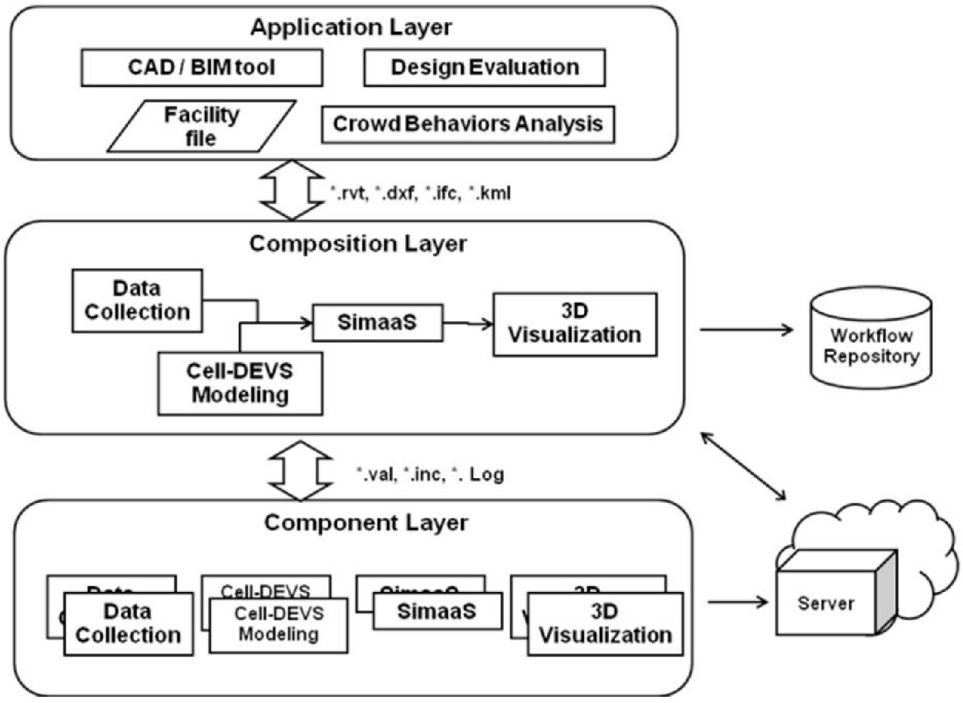


Fig. 2. Three-layer Workflow Architecture for Pedestrian M&S.

of developing and integrating varied M&S services and resources as mashups.

In particular, we are interested in building mashup applications for DEVS/Cell-DEVS related resources. DEVS [48] is a well-defined formal methodology that allows one to define hierarchical modular models. DEVS expresses a system as a number of connected behavioral (atomic) and structural (coupled) components. Cell-DEVS [40] is an extension to DEVS that defines each cell in a cellular model as an atomic DEVS. Each cell changes state according to the values of its neighbors following the rules defined by a local computing function. Numerous DEVS/Cell-DEVS models have been developed in different domains (communications, biology and medicine, architecture, defense, traffic, environment, etc. [40]). Developing DEVS/Cell-DEVS applications involve different resources: developing models, setting up simulation environments, collecting data from other resources, analyzing and visualizing simulation results. In addition, many Web APIs are useful for kinds of M&S application, e.g., weather forecast, Flickr pictures, YouTube videos.

3. The MAMS architecture

In this section, we introduce the MAMS architecture to develop M&S resources as boxes and integrate them as mashups. In an early effort, we introduced a workflow-based approach [27]. We will show a pedestrian M&S workflow as a case study. The workflow integrates different activities (e.g. data collection, cloud-based simulation, and visualizing results). In order to deal with these issues, we defined a workflow architecture introduced in Fig. 2. The components here have two categories: *pedestrian simulation services* and *pedestrian M&S tools*. The pedestrian simulation services are accessible via the Web. A workflow can invoke these simulation services directly. We use the Taverna *RESTful WS* module to call these services [14]. On the other hand, the M&S tools provide particular functions. Component providers save them in the Cloud.

The architecture presented in is organized in three layers:

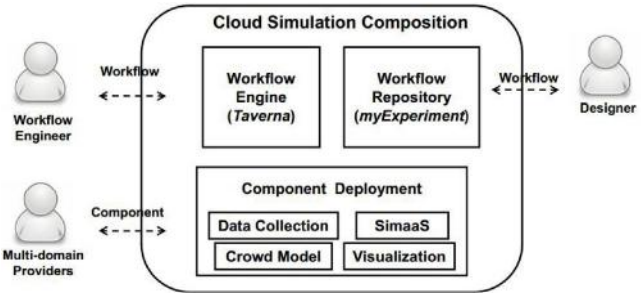


Fig. 3. The Composition Layer in the Cloud.

1) **The Component Layer** for deploying components in the Cloud. It consists of independent using well-defined interfaces and stored in the Cloud. The layer includes the following services:
   a) **Cell-DEVS modeling:** used for building the evacuation models using Cell-DEVS (various Cell-DEVS pedestrian models are available and can be expanded: bi-directional, etc.).
   b) **Data Collection:** used to extract data from the floor plans designed using BIM tools to a Domain Specific Model (DSM) instance, and use it as inputs for the simulation.
   c) Simulation as a Service (**SimaaS**): it is used to run simulations using web services.
   d) **3D Visualization:** includes a mechanism to parse simulation results and visualize them in BIM tools. This component can be customized.
2) **The Composition Layer** is responsible for defining workflows to formalize the integration of components and automate its execution. Users can replace components and modify the workflow for their particular purposes. In addition, these workflows can be stored in a workflow repository. Fig. 3 illustrates its overall structure.
   – **The Component Deployment** module it is responsible for deploying components in the Cloud. A component provides its functionality as an interface (a simulation service, a tool, a script, etc.). Its provider uploads and deploys it to the Cloud.

- **The Workflow Engine** is responsible for defining workflows. A workflow engineer is in charge of building workflows by linking components.
- **Workflow Repository**: it is responsible for sharing the workflows in the Cloud. Building designers can reuse a workflow, specify inputs and execute it using a workflow engine. In our case, we used *myExperiment* [13] as the workflow repository.

3) **The Application Layer** is responsible for collaboration. It allows building designers to run pedestrian models. The designers do not need to care about the workflow details: they choose a workflow and provide an input. Then, they wait for the end of the workflow and visualize the results. In addition, they can evaluate their design and restart the process.

Using this architecture, we can build workflows to execute like the one presented in Fig. 4. The workflow starts by taking a User Workspace, a Location, and a GIS Map as inputs, and produces, as output, a KML file to visualize the simulation results (which can be visualized on GoogleEarth). We first implemented the flooding models, using CD++ a GIS and weather data. The workflow produces the files needed by CD++ to start a simulation, as follows. The designer can provide these inputs, and run this workflow using Taverna, which will run the workflow and generate results and visualize them in Google Earth. We can share the workflow on myExperiment, e.g. this workflow can be found at http://www.myexperiment.org/workflows/2739.html)

Although this workflow approach has potential, we built our Mashups Architecture with Modeling and Simulation as a Service (MAMS), a more advanced approach that tries to simplify the development and integration process of M&S applications with heterogeneous data and services.

The MAMS architecture, presented in Fig. 5, is organized in five **Layers**.

1) **Cloud:** it is responsible for supporting the Cloud infrastructure. This includes *compute* units (for building and executing simulations) and *storage* units (for sharing M&S resources). This layer is also responsible for deploying M&S resources in the Cloud by using the *MSaaS middleware*, a platform for development, deployment and management of MSaaS.
2) **Box:** this layer allows developing M&S resources as mashup components (called *Boxes*). Each Box is identified by a uniform signature; also, they have their own function for handling input messages and their own visual form. There are different Boxes categories (e.g., MSaaS from the Cloud Layer, existing open WebAPIs, widgets, and operators).
3) **Tag Ontology:** this layer maintains and learns a tag-tree ontology for the Boxes using tag-mining and ontology-learning algorithms. We can mine tag signatures from the boxes; or use ontology-learning algorithms to construct domain-specific ontologies from the boxes tag signatures.
4) **Wiring:** this layer is in charge of connecting boxes and forming a mashup. Boxes can be linked with each other through their inputs/outputs. The same box can be reused and re-wired in different mashups. This layer can also work with the Tag Ontology Layer for semantic selection: a tag-tree built automatically by layer 3 can suggest "wirable" (composable) boxes.
5) **Mashup Application:** it is responsible to select and wire boxes as mashups in workspaces for particular M&S requirements, and run mashups at run time. It provides a user-friendly GUI to manage and select boxes, wire them, execute and visualize them at run time.

The contributions of this MAMS architecture can be summarized as follows:

- MAMS simplifies M&S development and integration, using MSaaS to make *everything as a service*. It uses a simplified lifecycle to develop, deploy, identify, select, integrate and execute varied M&S resources as services in the Cloud.
- MAMS is the first existing Mashup development process for M&S. Following a new Box/Wiring/Mashup method, users can develop resources as mashup components, compose them as mashup and run these mashups in web browsers quickly.

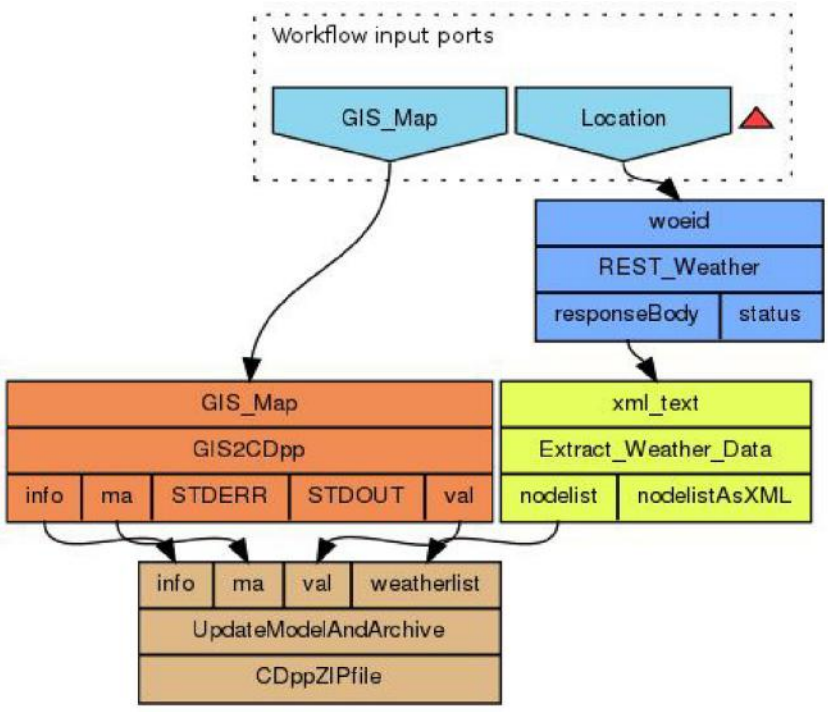


Fig. 4. Overall workflow in Taverna.

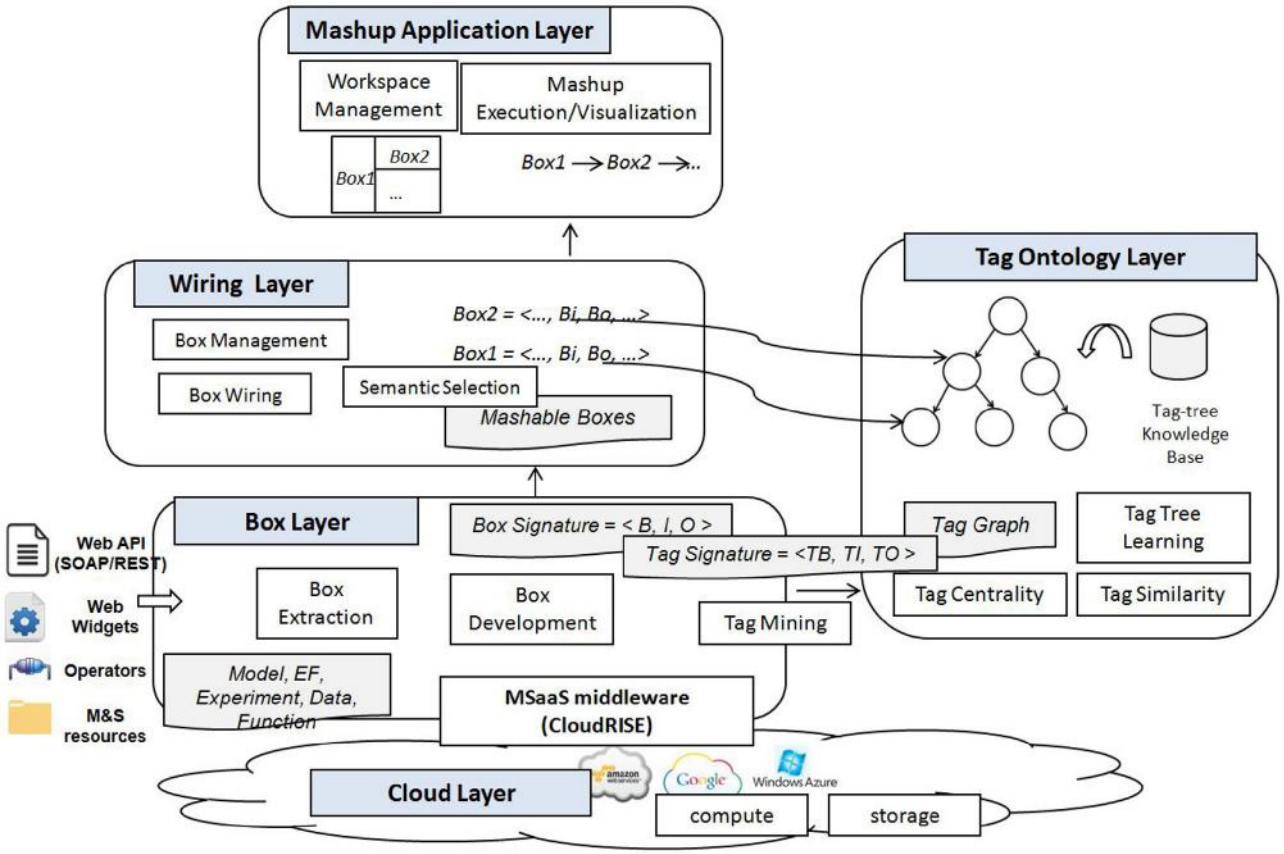


Fig. 5. Mashup Architecture with Modeling and Simulation as a Service [45].

- MAMS helps collaborative development from different experts (e.g., Cloud Administrators, Box Providers, Ontology Managers, Users, etc.). Each person can focus on a specific activity, hide technical details in boxes and reuse them in mashups.

The Cloud layer and the MSaaS middleware [43] provide simplified APIs of the underlying Cloud infrastructure, which facilitates access to the Cloud. CloudRISE exposes M&S resources as services, using RESTful WSs. In addition, this layer supports a VM-like *MS-Image* method for building experimentation environments. The Tag Ontology layer [42] simplifies the selection/wiring process using semantic selection with data mining and machine learning algorithms. We designed an automatic identification method to add semantic to each resource using tag-mining techniques. We also designed a new algorithm to "learn" a tag-tree ontology from mined tags, avoiding predefined ontologies and helping to automate the M&S application development [44].

The MAMS architecture improves our previously defined workflow approach as it provides a different way to develop and integrate M&S applications, summarized as follows:

1. MAMS integrates resources at the web service level. When a service is ready, we can put it in a box and use it directly. In contrast, the workflow only uses SimaaS, which is at a lower level of abstraction as it only focuses on simulation. Functions are deployed in the Cloud, and workflow calls them directly by system-level commands.
2. MAMS provides a view for each box, facilitating visualization. In contrast, workflows focus on the integration of components (no visualization of components at run-time).
3. MAMS integrates all kinds of M&S resources, providing easy ways to build run-time mashups. In contrast, the components in workflows are a subset of the resources available (i.e., SimaaS and functions components, but not Widgets, WebAPIs or Operators).

In the following sections, we will provide a detail discussion about the top layers of the architecture [45], which focuses on the development of mashup applications.

4. M&S mashup method

The M&S Mashup method is based on "wirable" boxes, dealt by the Box, Wiring and Mashup Application Layers. Together, they support a lightweight life cycle for developing M&S applications. In Fig. 1, we showed a standard mashup development process [10]. Fig. 6 shows M&S Mashups as a special version of this process.

The M&S Mashup life cycle can be divided in the following steps:

1) **Box development and identification** (implements *Discovery and selection*). Boxes are the fundamental elements of the MASM mashups method. They cover all the resources and services. Boxes can be discovered and selected using their universal structure (the *Box Signature*), which identifies inputs, outputs and functions.
2) **Box wiring** (defines the *Mashup composition* process). This mechanism supports mashup composition. Boxes can be linked with each other by their input/output ports identified in their Box Signatures. After wiring, a mashup is developed and can be used in the next process.
3) **Mashup execution and workspace** (implements the *Usage and maintenance* process). The mashup platform supports the management of the workspace, which deals with using and maintaining the mashups. Users can run the mashup in the

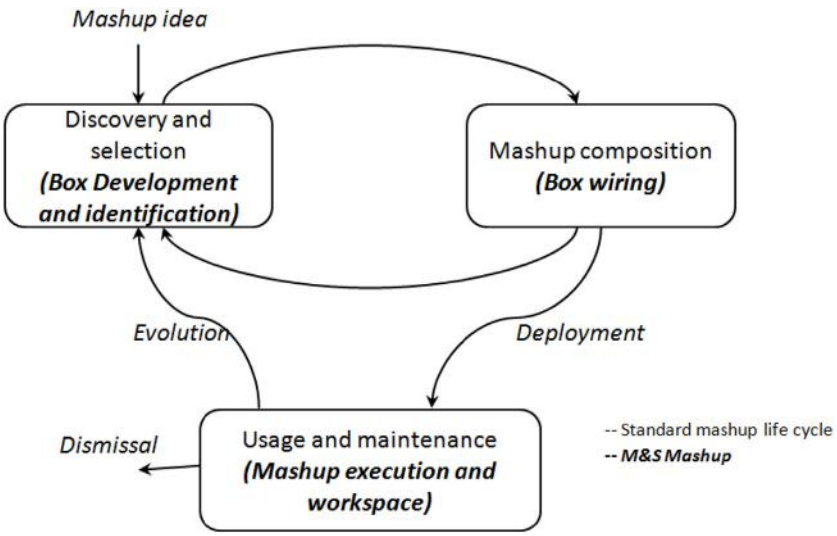


Fig. 6. Mappings of M&S mashup and the standard mashup life cycle.

workspace and visualize the results (e.g., simulation results, runtime logs).

Overall, the M&S Mashup method is prototype-centric and iterative; it allows continuous improvement and evolutions. An M&S mashup can be run by different users at different times. If the results are not satisfactory, they can modify the mashup by updating and re-wiring boxes. In the following sections, we will discuss each part of the process in detail. Some of the contents in these subsections have been presented in [45]; the reader can find further details there.

4.1. Box – developing mashup components

Boxes represent mashup components for M&S that can be executed on web browsers (e.g., fetching a model, running a simulation, visualizing results, etc.). They receive data from varied services, transform the data and send new events to others. There are four basic box types:

- **MSaaS**: it uses CloudRISE, which exposes M&S resources as Restful WS (accessible via the HTTP methods GET/PUT/POST/DELETE to the corresponding URIs). The MSaaS Boxes allow combining different simulations (each on a different box).
- **WebAPI**: it uses existing Web APIs, using RESTful or SOAP-based WS (we can use WADL but in many cases, the HTML pages describe their REST APIs).
- **Widget**: it is a lightweight web application to show the data on web browsers, and it can represent data visually. It can be developed using HTML/CSS/JS or it can be reused from the Web.
- **Operator**: it takes input data from other boxes, and it generates outputs. We provide operations to deal with inconsistencies between boxes. They can be used as filters, aggregators, splitters, or adaptors. Operators are not viewed in the web browsers.

Each box package has three parts:

- **Box Signature**: it identifies each box with basic information (name, type, subtype, author, URI, WS method and description), and input/output ports (with port name, type and description). Box Signatures can be described in XML files or built automatically from existing sources.
- **Box Function**: each box has a function to respond to input events. Different types of boxes have different functions: the MSaaS Box combines multiple MSaaSs for a given experiment; the WebAPI Box executes the function defined in the API; the Widget Box handles visualization; the Operator Box executes an action (e.g., splitting, combing, and data conversion).
- **Box View**: Each box can also be optionally viewed in web browsers. For MSaaS and WebAPI boxes, their views could be their signature or execution status. Users can customize the view.

4.2. Box wiring – linking boxes

Boxes can be connected to each other, allowing the composition of different boxes through. Each Wiring contains a set of boxes and connections. Box Wiring is based on an Event-driven architecture (EDA). Boxes can interact with each other via events (i.e., anything that is "change in state"; for example, after a simulation finishes, the simulation changes from "running" to "done"). The M&S mashup transmits events among loosely coupled boxes. A wire allows the transmission of events from emitter boxes to consumer boxes. An Emitter Box has the responsibility to generate events, it does not know the consumers of the event, it does not even know if a consumer exists, and in case it exists, it does not know how the event is used or further processed. A Consumer Box has to react as soon as an event is presented. For instance, an MSaaS box or WebAPI box will call services provided by CloudRISE or Web APIs; a widget box will provide a view for the receiving events; and an operator box will filter, transform or simply forward the event to other boxes.

The Box wiring implies is a general approach to compose resources. There is no constraint about the logic flow among the boxes. The wiring among boxes supports a loosely coupled sequential composition. Operators can help to improve this composition with complex controls (e.g. filters, aggregators, splitters, adapters, BPM rules). In addition, boxes can contain heterogeneous data and they can be linked together.

4.3. M&S mashup – building mashup applications

The boxes and wiring defined in the previous sections can be used to build *M&S mashups*, consisting of a set of Boxes, Wirings, and a User Workspace. In the following sections, we will explain how MASM implements these mashups and will discuss a prototype implementation.

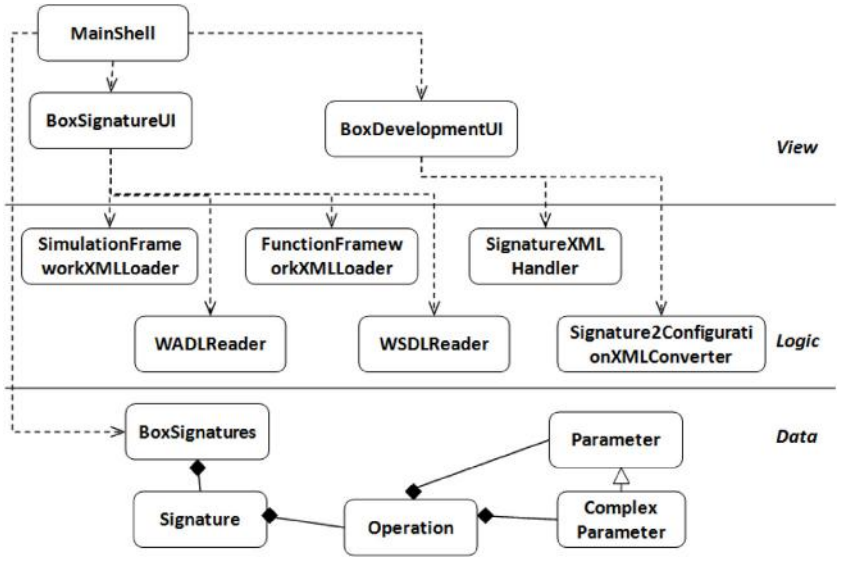


Fig. 7. Class Diagram of Box Development Tool.

5. M&S mashup implementation in the MASM architecture

In this section, we discuss the Box Development Tool (that is used to extract and develop boxes) and the Mashup Platform (that is used to wire boxes and run M&S mashups).

5.1. Box development

In [45], we introduced the definition of a Box Development framework that can be used by users to load and save Box Signatures in XML. The framework can also extract Box Signatures from existing files. In addition, it can generate a configuration XML file used in the box package, and it can suggest similar existing boxes to the users (Fig. 7).

Boxes are developed using web technologies (XML, JS, HTML/CSS and the WireCloud environment). A Box Signature is an XML configuration file, associated to a Box Function (a JS file that defines the actions for input events) and a Box View (HTML/CSS files to show the data in web browsers). For each box, we generate its configuration as XML. The Box Function is a JS file that defines the actions for input events, while the Box View contains HTML/CSS files to show the data in web browsers. For each box, we generate its configuration as XML. For the JS and HTML/CSS files, this Box Development Tool can suggest to users similar boxes.

The JS function triggers inputs events by reusing the WireCloud's API: MashupPlatform.wiring.registerCallback(inputName, callback). When the function finishes, it outputs message by reusing the WireCloud's API: MashupPlatform.wiring.pushEvent(outputName, data). The JS function can execute SOAP WS or RESTful WS. In general, executing these WSs need to build input messages, ports, and handle output messages.

1) For RESTful WS, the input message is the input event, the channel is the HTTP method (GET/PUT/POST/DELETE) on a particular URL, and the output message is the HTTP response either in XML or JSON format.
2) For SOAP WS, the input message is a SOAP message, the channel is the HTTP POST method, and the output message is sent in the HTTP response.

Fig. 8 shows the GUI of Box Development Tool that extracts the Box Signature from the Facebook REST APIs. As we can see, different APIs were extracted from Facebook's WADL file. Each API contains an operation name, method, description, inputs and outputs. The API searches public objects (e.g. user, photo, page, etc.). We can see the API with its *method* (GET-search), its *description* (Search over all public objects in the social graph), the *inputs* (two parameters: a *query string* and an *object id*), and the *outputs* (search results). For the WebAPIs not supported by WADL/WSDL files, they can be obtained from online HTML pages. In addition, the user can build a signature XML. Such services can be found online (some popular service repositories in HTML are https://www.mashape.com/explore and http://www.programmableweb.com/).

Fig. 9 shows an example of the generation of configuration files for box packages. After loading and extracting the box signatures, a user can select an API from the GUI and save it to the configuration format used in the M&S mashup platform. The figure shows a REST API found at www.mashape.com to convert geographical information (e.g., *latitude* and *longitude*) to address-related outputs (e.g., *city* and *zipcode*). In addition, the Box Development Tool can suggest users with similar box package samples based on the box type, so users can reuse them for fast box development. For example, in the figure the user can load saved signatures, select the *geoIpToAddress* REST API, and specify a file-path.

5.2. M&S mashup platform in the MASM architecture

Our M&S Mashup Platform for the MASM architecture includes a wiring editor to build M&S mashups rapidly, was built by extending WireCloud. WireCloud is an open source application that supports widgets, wiring, user workspace management, and mashup execution, but it does not support different types of Boxes. We extended the tool to define new methods for supporting the MASM architecture. Some of the details have been included in the Appendix A. We used Ajax provided by JQuery to box wiring (M&S mashups can send data among boxes asynchronously). Boxes are developed in HTML/JS/CSS. We implemented a mashup platform to store all boxes and define all wirings. Events are transmitted between boxes as XMLHttpRequest objects. Each box may have input functions (for handling received events) and output functions (for sending events to other boxes). The M&S Mashup Platform is implemented in Python using Django, an open-source web application framework (the tool used to build WireCloud). Django supports the MVC

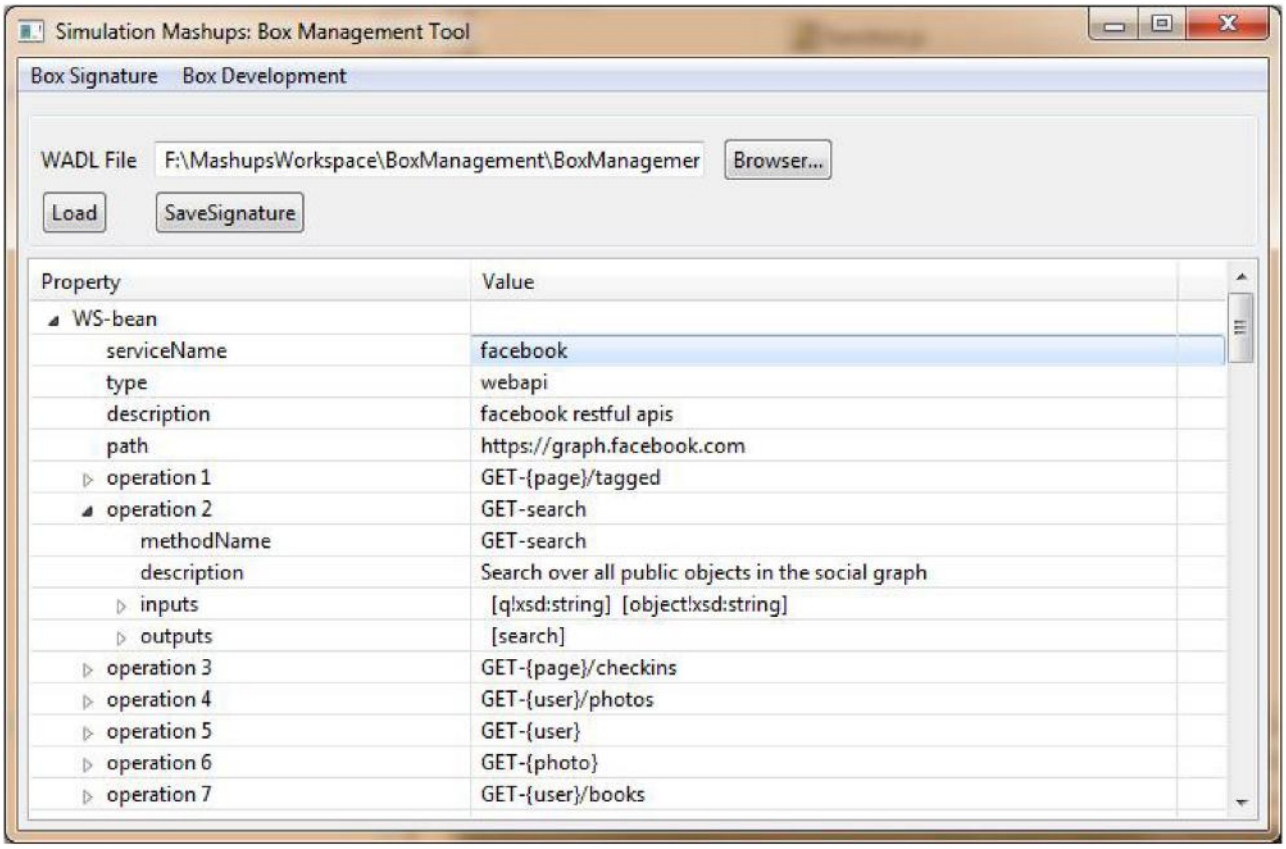


Fig. 8. Extraction Box Signature from Facebook REST WebAPIs.

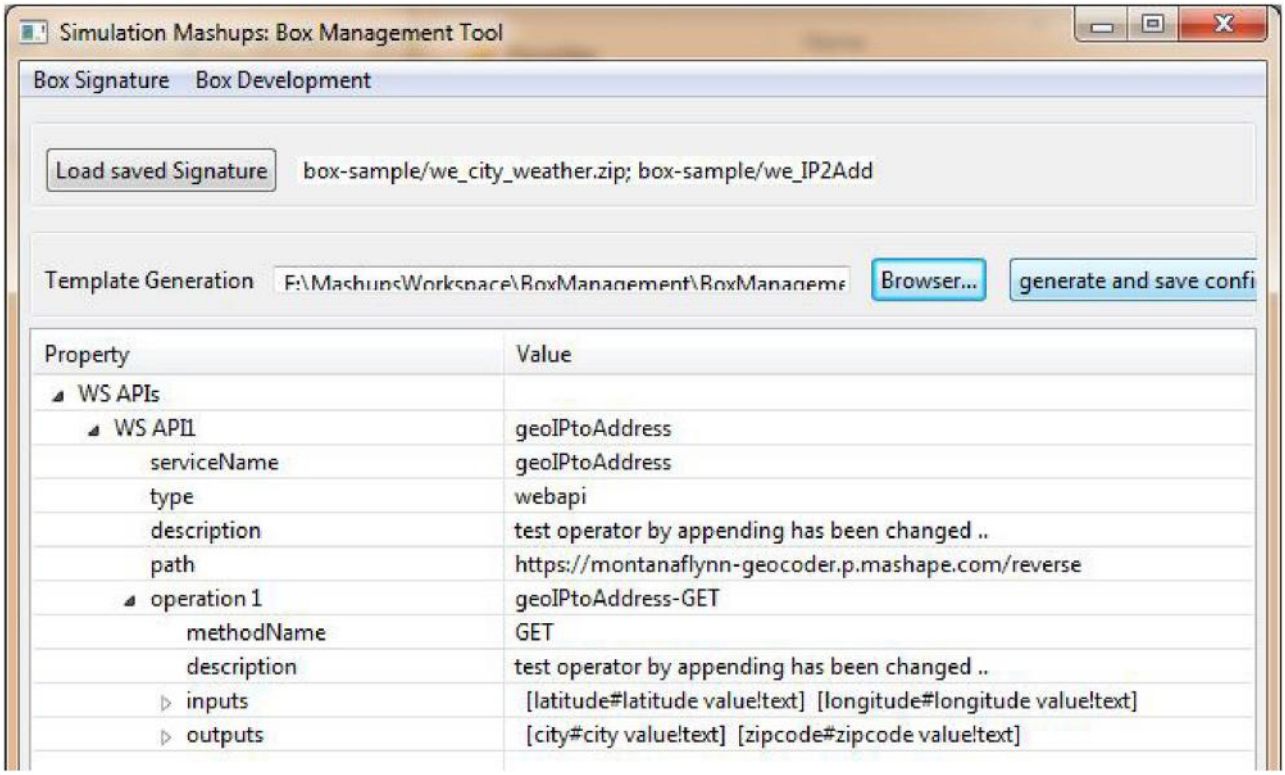


Fig. 9. Generating Box configuration and suggesting similar Box Packages.

Fig. 10. Box Search and Uploading.

(Model-View-Controller) architectural pattern, allowing building complex database-driven websites.

At first, the platform can manage boxes by searching and uploading them, as shown in Fig. 10. Users can choose the box zip file developed from the above Box Development Tool, and upload it into the M&S Mashup Platform.

Fig. 11(a) shows the box selection page. The platform will keep the boxes that user uploaded. Users can either search keywords of box by their name, or select particular type of box, and then the mashup platform will return users with matched boxes. Fig. 11(b) shows how to add box resources into a user workspace and customizing its layout.

After the boxes are loaded, users can select them, then drag and drop them into the workspace. Fig. 12 shows how to wire boxes. In this example, one box is a widget that allows users to provide a GIS TIFF URL (containing GIS dataset in a particular area); another box collects information from a GIS TIFF file and generates initial values for simulation model. After wiring the output port "keyword" of GIS TIFF box and the input port "GISTIFF" of GIS data collection; the GIS TIFF data will transmit between them.

After this step, the mashup is ready and users can visualize it at run-time. Fig. 13 shows how the mashup looks like for the above example. The user can see the workspace page, and provide input values for "GIS-TIFF URL". After pressing the "send" button, the GIS data collection box will receive the input. In that box, data is collected, and corresponding results are shown (e.g., data has been successfully collected).

The M&S mashup platform is deployed in the Cloud. The current implementation is in AWS. We also saved all the configuration of this platform in an AWS image, so we can recover it easily when the platform is down. Like CloudRISE, we can choose different instance type (e.g. free, high memory, high CPU) for the mashup platform.

Fig. 12. Box wiring.

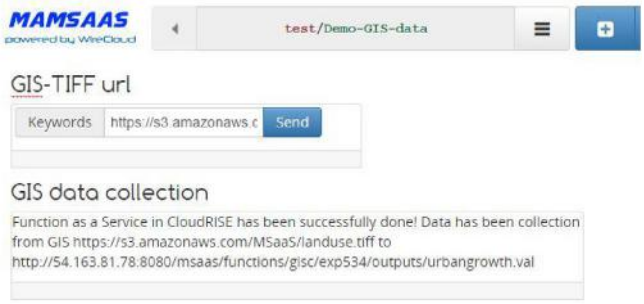


Fig. 13. Mashup execution.

6. Case studies: M&S mashup applications

In this section, we will show several M&S mashup cases, to demonstrate how our proposed architecture and developed tools can simplify the development and integration of M&S application.

6.1. A cell-DEVS forest fire M&S mashup

Fire spreading is a complex phenomenon, which depends on many variables, including the type of fuel, the topology of the area, the weather, etc. M&S technologies have been used to study forest fires, and M&S is now generally the preferred solution for predicting the behavior of wildfires. However, building such applications involves complex models, and configuring simulation environments to run varied model experiments. Likewise, many open APIs can be helpful to improve such application (e.g., fire news, YouTube videos, etc.). In order to show our method, in this section we present a case study for a forest fire simulator using a mashup in the MAMS architecture.

In [39], we presented different Cell-DEVS models to study the forest fires spreading in forest. Here, we use one of these Cell-DEVS model as example to show how to build the mashup. This application uses many resources, including:

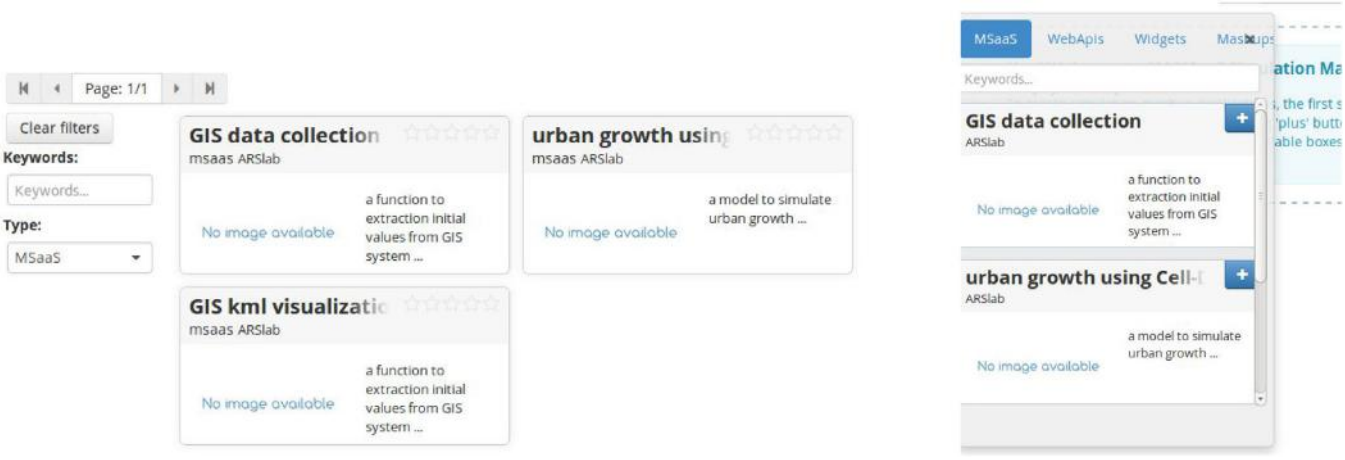


Fig. 11. (a) Box selection (b) Adding box resources into user's workspace.

- Input model: a Cell-DEVS forest fire model to simulate different scenarios.
- Cell-DEVS simulation: simulation experiments for the forest fire model in the Cloud.
- Results analysis: a function to analyze simulation results.
- Results visualization: a widget to visualize the forest fire simulation.
- WebAPIs: useful for studying Forest Fires (e.g., search the latest forest fires news).
- YouTube videos: a widget to show videos related to forest fire simulations.
- Operators: for simplifying the developing process, such as searching existing supported files based on given forest fire models, or obtaining log files from simulation runs.

We used the M&S mashup method and the MAMS architecture to develop the application. Following the layers of MAMS, the general development process of building this mashup is:

1. to develop the MSaaS services
2. to develop box components for each resource
3. to wire the boxes in a mashup
4. to run and execute the mashup

The first step is to develop resources as MSaaS services. Different users may have resources that are only accessible in their computers locally. For example, they may have high quality models that are not shared; likewise, the simulation environment may be complex to set up. In this first step, we develop these resources as Services. To do so, we use the CloudRISE middleware (see Section 3). For instance, the model/data/simulation/function APIs support Model, Data, Experiment, and Function *as a Service*. Using this layer, we share a model to an URI, for instance https://{cloud-rise-instance}/msaas/models/cell-devs/firespreading. Here, {cloud-rise-instance} refers to the IP address of the computer running CloudRISE (which can run in different computers, for instance, our servers at Carleton University, or Amazon EC2). For experiments, we can keep the Experimental Framework (EF) for the model in a different URI, for instance, https://{cloud-rise-instance}/msaas/simulations/firespreading. We can use this service to control the simulation life cycle (e.g., configuring/starting the experiments, checking the execution, or retrieving results).

Then, we package each resource as box (as discussed in Section 4.1, the basic component of a mashup is box, with signature, function and view). The boxes for this case study are as follows:

- **MSaaS** uses the services developed in the first step. In this case, we have boxes *Input Model* and *Cell-DEVS simulation*. Their signatures are extracted from the configuration files from CloudRISE; their functions call MSaaS services and their views show their execution status.
- **WebAPI** calls an existing open Web API. In this case, we use *Web search* (a REST WS to search inputs from web pages like Google, Yahoo and Bing, https://market.mashape.com/faroo/faroo-web-search). Using the Box development tool, we can load its box signature; also, its function calls the Web API; and the view shows the search result.
- **Widget** shows data in web browsers. In this case, we have two widgets: *Log viewer* (a web viewer for Cell-DEVS simulations) and *YouTube videos* (a YouTube plug-in).
- **Operator** handles inconsistencies between boxes. In this case, we use *get all files from model* (which obtains existing support files for a given model) and *get log file from zip* (which extracts simulation logs from archives).

All boxes have similar structure: box signature, function and view. Here we show an example for a web search WebAPI Box that uses an open API provided by Faroo, which can search news and articles from more than 2 billion pages (http://www.faroo.com/hp/api/api.html). This API is integrated from Google, Yahoo and Bing. It has been used by more than 3700 third-party applications. At first, we use the Box Development Tool extract a configuration file as follows.

```
<?xml version='1.0' encoding='UTF-8'?>
  <webapi xmlns="http://wirecloud.conwet.fi.upm.es/ns/macdescription/1" vendor="ARSlab"
                                    name="WebSearch" version="1.0">
    <details>
      <title>Web Search</title>    <authors>Sixuan Wang</authors>
      <description>News search from more than 2 billion pages </description> …
    </details>
    <contents src="index.html" useplatformstyle="true"/>
    <rendering width="5" height="20"/>
    <wiring>
      <inputendpoint name="keyword"  type="text"  description="keyword"…/>
    </wiring>
  </webapi>
```

This configuration file specifies its *vendor* (ARSlab), *name* (WebSearch) and *version* (1.0). In addition, it stores the *author*, *description* and *view page* (index.html), as well as *ports* information (input port "keyword"); note that this box does not have an output port. Next, we can see the view page (index.html), which includes several JS definitions (*JQuery* for the AJAX API call, *xml2json* for converting XML to JSON, *function.js* for the customized box function). It also has a test field "textarea" to show the box content.

Finally, the *function.js* file specifies the function when an input comes. In general, MashupPlatform.wiring.registerCallback ('keyword', inputfunction) defines the input handling function "inputfunction". This function calls the web search API with following parameters: the URL https://faroo-faroo-web-search.p.mashape.com/api and the input model name. Then it parses result and show its web content (e.g. titles and links) in "textarea" of the page view.

```
<html>
  <head>
    <script type='text/javascript' src='https://code.jquery.com/jquery-git.js'></script>
    <script type='text/javascript' src="https://x2js.googlecode.com/hg/xml2json.js"></script>
    <script type="text/javascript" src="js/function.js"></script>
  </head>
  <body>              <div id="textarea"></div>              </body>
</html>
```

Finally, we define the function.js, which is activated upon inputs. In general, *MashupPlatform.wiring.registerCallback ('keyword', inputfunction)* defines the input handling function "inputfunction", which calls the web search API with the URL https://faroo-faroo-web-search.p.mashape.com/api and the input model name. Then it parses result and show its web content (e.g. titles and links) in "textarea" of the page view.

```
(function () {
     var inputfunction = function inputfunction(event_data) {
         var div = document.getElementById("textarea"); …
         generalRestGet(modelname, function (results) {
              var j_results = JSON.parse(results);
              for(i=0; i< j_results['results'].length; i++) {
                finalResult +=    JSON.stringify(j_results['results'][i]['title'])+
                             JSON.stringify(j_results['results'][i]['URL']) ;     }

              div.innerText = 'web search results from Google, Yahoo and Bing for (' +
                             modelname + ' spreading):\n\n'+finalResult;
         },…    };
        var generalRestGet = function generalRestGet(word, onSuccess, onError) {
           var URL = 'https://faroo-faroo-web-search.p.mashape.com/api?q=' + word;
           MashupPlatform.http.makeRequest(URL, { method: 'GET',
            "requestHeaders": {… "Accept": "application/json"   },
            onSuccess: function (response)    { onSuccess(response.responseText); },
            onError: function () {  …   }
         });  };
        MashupPlatform.wiring.registerCallback('keyword',inputfunction);
})();
```

After the boxes are developed, user can package it as a zip file and upload it to the M&S Mashup Platform. Fig. 14 shows the uploaded boxes in the mashup platform.

Once the boxes are complete, we can wire them into a mashup. Fig. 15 shows the box-wiring page in the M&S Mashup Platform. We can wire the output of *Input Box* and the input of Operator *get all files from model*, which will output different supporting files such as the *model* (the input model URI), *initialization* (the initial configuration values for this model), and *palette* (to choose colors to visualize the results). In addition, we can wire the outputs of *model* and *initialization* as inputs for the *Cell-DEVS model simulation* in order to create a new Cell-DEVS experiment. After, we can wire the output port of *Cell-DEVS model simulation* to the Operator Box *get log file from zip* to extract the log file, and then wire its output *log* to the input *result of log viewer* for visualization. In another path, the *Input Box* is connected to the WebAPI Box *Web Search* (to get the latest posts related to the given model name) and the Widget Box *YouTube* (to get related videos).

After this step, this mashup is ready. Users can execute and view it in their workspaces in the M&S Mashup Platform. Fig. 16 shows an example of a Forest Fire simulation. After entering a Forest Fire Model URI, the M&S mashup will run. Each box executes its function when receiving input events and shows the corresponding visualization. We can see that the *Cell-DEVS simulation* Box has been executed successfully, and a Forest Fire viewer was generated and shown in the right part of Fig. 16. In addition, this mashup also shows the information of web search results, in which it queries the latest posts with the keyword "Forest Fire". Related YouTube videos about "Forest Fire" are also displayed.

Although focusing on forest fire simulation, this generic mashup can be reused for other Cell-DEVS models by only changing the input model URL. Fig. 17a shows a mashup example for a model focusing on snowflake formation. Fig. 17b shows another mashup example for the interaction of cancer and the immune system. For each mashup, we can see the simulation results, the web search and the videos information. These modifications require minimum changes in order to build applications adapted to a given application domain, as seen in the figure.

This wiring process can be simplified using the semantic selection approach in MAMS. As discussed in [42,44], this approach can help the wiring process by suggesting "wirable" boxes. First, we get tags from box signatures. Then we build a tag-tree as domain-specific ontology (this tree can be learned from the tags or provided by ontology experts). The "wirable" boxes are based on their tags and this tree. For example, if Box A has an output with tag a and Box B requires an input with tag b, and a is the child of b in a tag-tree (noted as $a < b$), we say that Box A is wirable to Box B. Fig. 18 shows

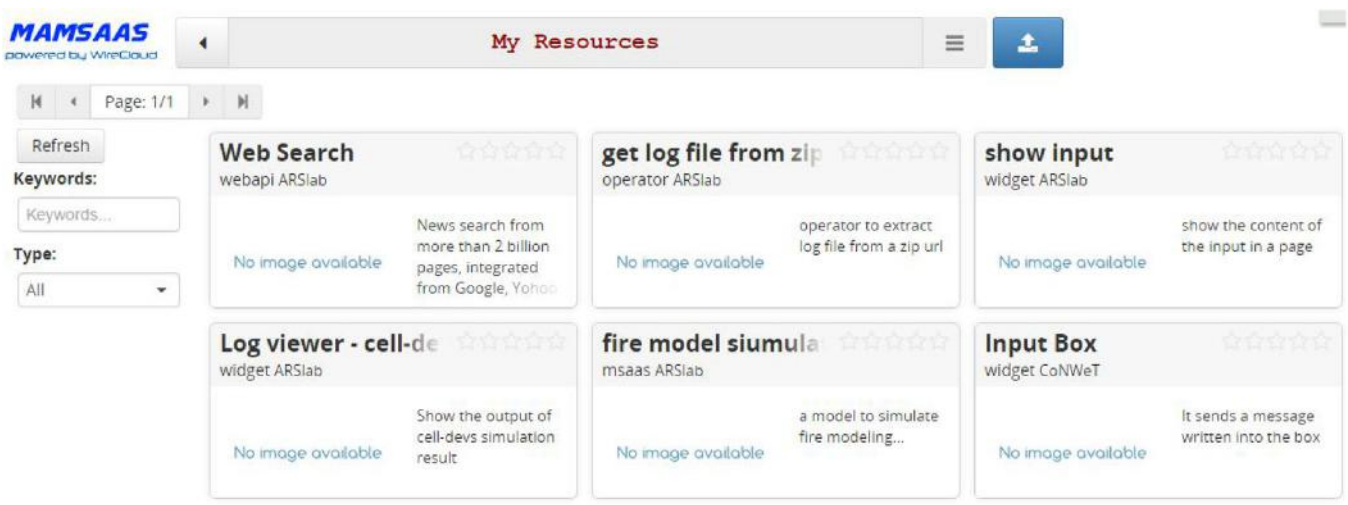


Fig. 14. Uploaded boxes for Fire Spreading M&S mashup.

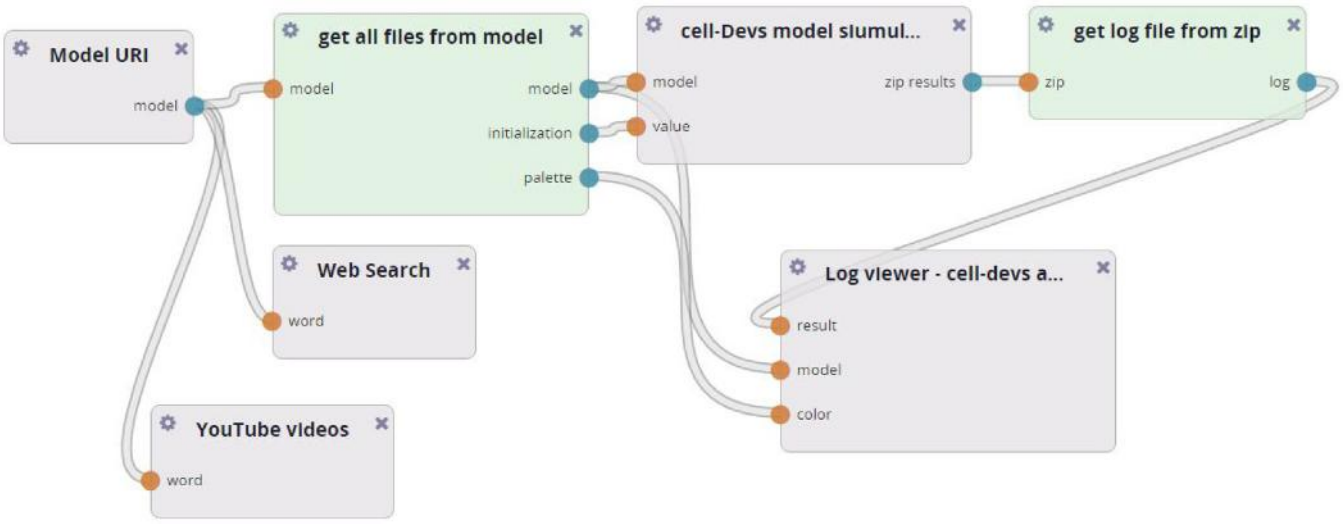


Fig. 15. Boxes wiring for Forest Fire M&S Mashup.

parts of the tag-tree used in this Cell-DEVS mashup. We can see several sub-tag relations: *simulation* < *model*, *initialization* < *value*, and *palette* < *color*.

For this case, we can simplify the wiring process base on the "wirable" boxes. Assume that a mashup engineer is not sure which boxes to use; s/he is only sure about *Cell-DEVS model simulation*. This Box has two input ports (*model* and *value*), and one output port (*zip results*). By searching *model* and *value* as output tags, the semantic selection algorithm can suggest Boxes with those ports, and the mashup engineer can select the one that meets their demands.

For instance, it can suggest the Box *get all files from model* since *initialization* < *value* and *model* = *model*. Similarly, when searching *zip result* as output port, since *get log file from zip* also has an input *zip*, *Cell-DEVS model simulation* can wire to this box. Since *log* < *result*, *get log file from zip* can link to *Log viewer*. Since *palette* < *color*, *get all files from model* can link to *Log viewer*.

6.2. A sample DEVS M&S mashup

We can also use MAMS to develop mashups reusing basic DEVS models. We illustrate such services using the Barber Shop model, a traditional queuing model that covers the basic logic of queuing and processing of customers. We assume there are limited barbers, and customers are served First-Come-First-Serve. Customers might need to wait for their turn if all barbers are busy.

For developing MSaaS services, we reuse the *model* service and the *simulation experiment* service discussed in the previous mashups, as the CD++ simulator used in the experiment (running at the lower layers of the architecture, driven by CloudRISE), can be invoked to receive any DEVS/Cell-DEVS models to be executed remotely. In this case, we modified the Cell-DEVS model in the model service, and replaced it with a DEVS barbershop model we defined.

Then, we developed new boxes using these services, as follows:

- MSaaS Box: a *DEVS model* of the barbershop, and a *DEVS model simulation* to run the simulation.
- WebAPI Box: a *Web Search* to search inputs from web pages like Google, Yahoo and Bing; and a *Publication Search* to obtain related publications.
- Widget Box: *Wikipedia* to show related information, and *Get content* to show the results.

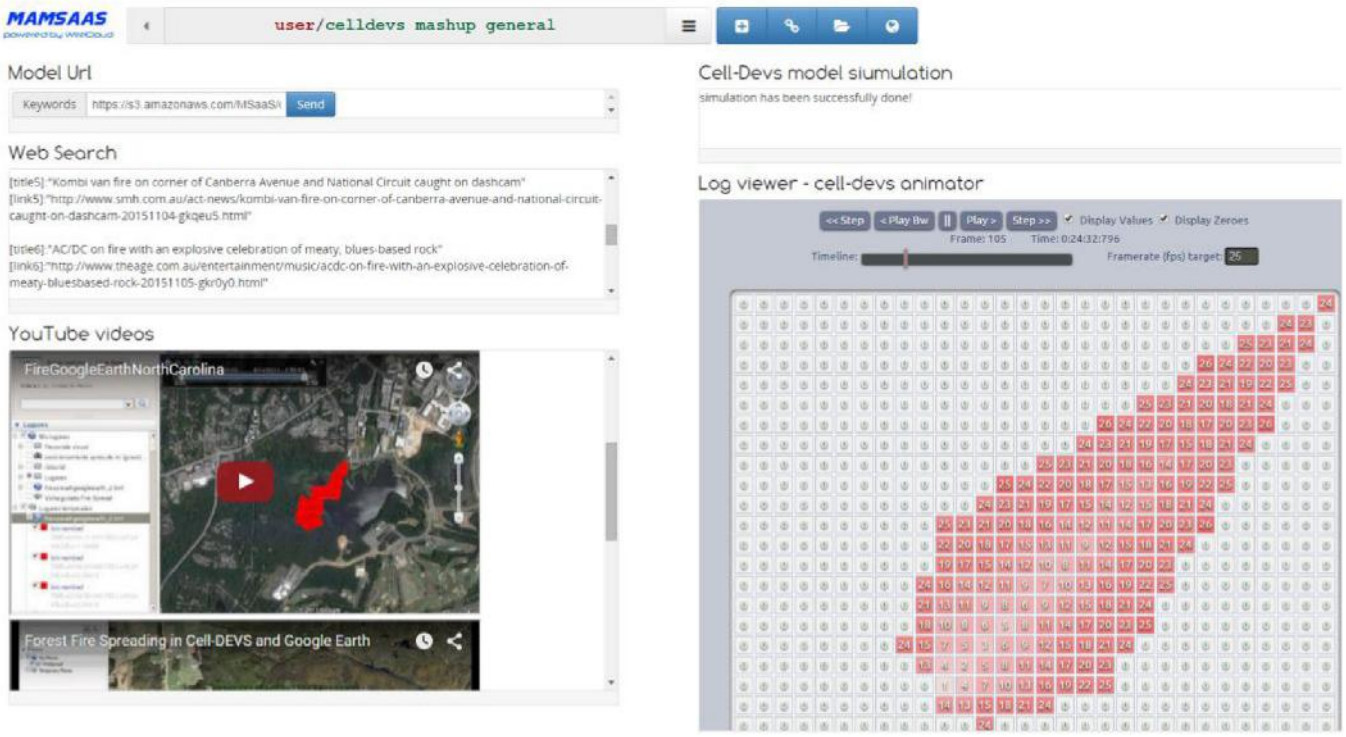


Fig. 16. Forest Fire M&S Mashup application.

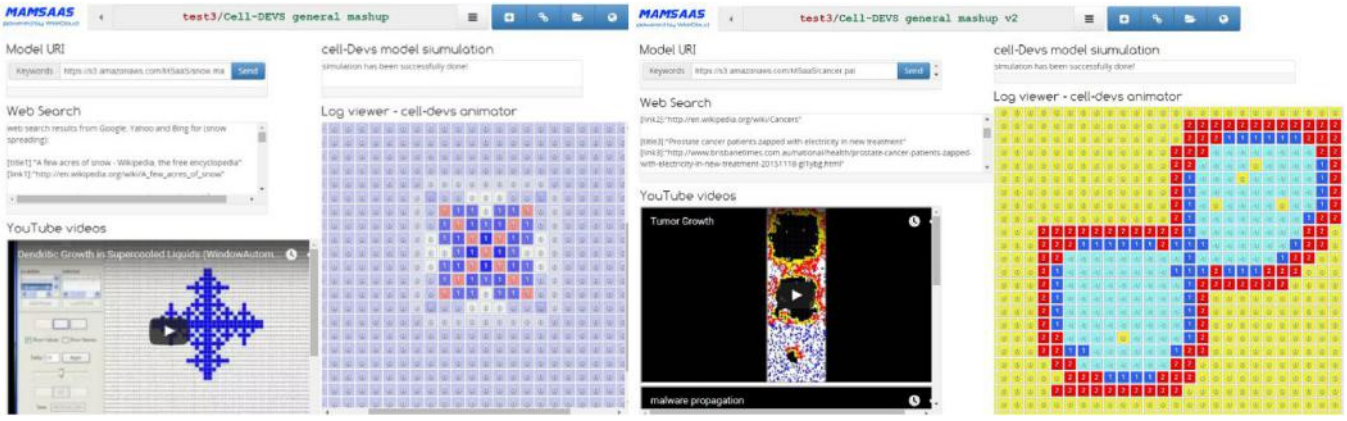


Fig. 17. a) Snowflake formation Mashup b) Cancer/Immune system Mashup.

- Operator Box: *Get files from zip* to obtain files and *Fetch model name* to extract model names from a given URI.

When we wire these boxes into a mashup, we obtain the box-wiring in Fig. 19. As we can see, we use one input: a *DEVS model*. Then, we connect it to the *DEVS model simulation* (running in CloudRISE). Then, the results are connected to *Get files from zip*. In addition, the model also is connected to *Web Search*, *Publication Search*, and *Fetch model name*, connected to Wikipedia.

After the boxes have been wired, the mashup is ready. Fig. 20 shows its execution. We can see the simulation results and we can study the simulation logs. In addition, different publications related to the Barbershop model, wiki information and other web information are displayed.

This mashup has combined: a DEVS model (to simulate the barbershop scenarios) that can be executed using Cloud-based Simulation (and CloudRISE), an operator to obtain the simulation results files, a widget for web browsers to visualize the simulation result (in this case, a simulation log file with all the output messages generated by the simulation), a Wikipedia service call, a Web Search for related articles, and a list of existing publications.

6.3. A Building Evacuation Mashup

This section shows a complex mashup example for studying pedestrian behavior during building evacuation. In recent years, the population in public areas and transportation facilities has become much denser, thus, predicting and trying to control the behavior of pedestrians is an important issue. M&S can support such analysis and it can help designers to evaluate the performance of their designs. Designers can use Pedestrian M&S to find flaws in particular areas of their buildings or urban area designs before the construction has begun.

To better study the behavior of a pedestrian in the design of a building or city section, we need layout information. Nowadays, new technologies and tools are available to manage this

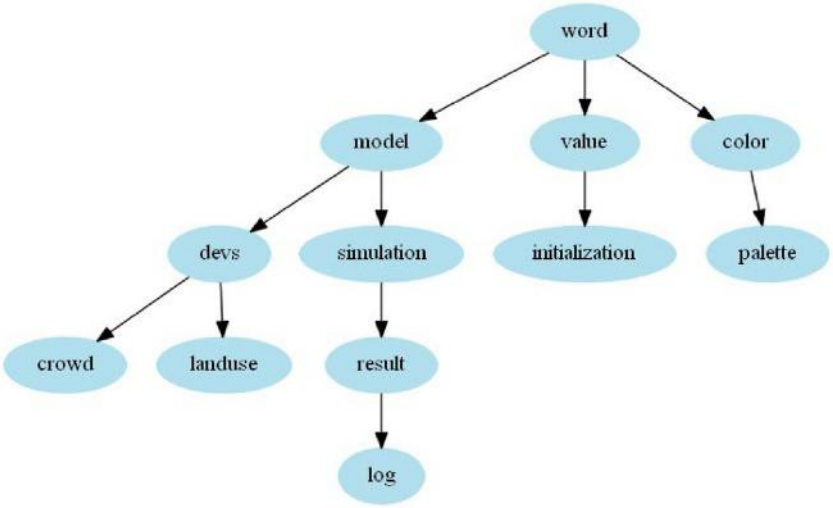


Fig. 18. Part of Tag-tree used in Forest Fire M&S Mashup.

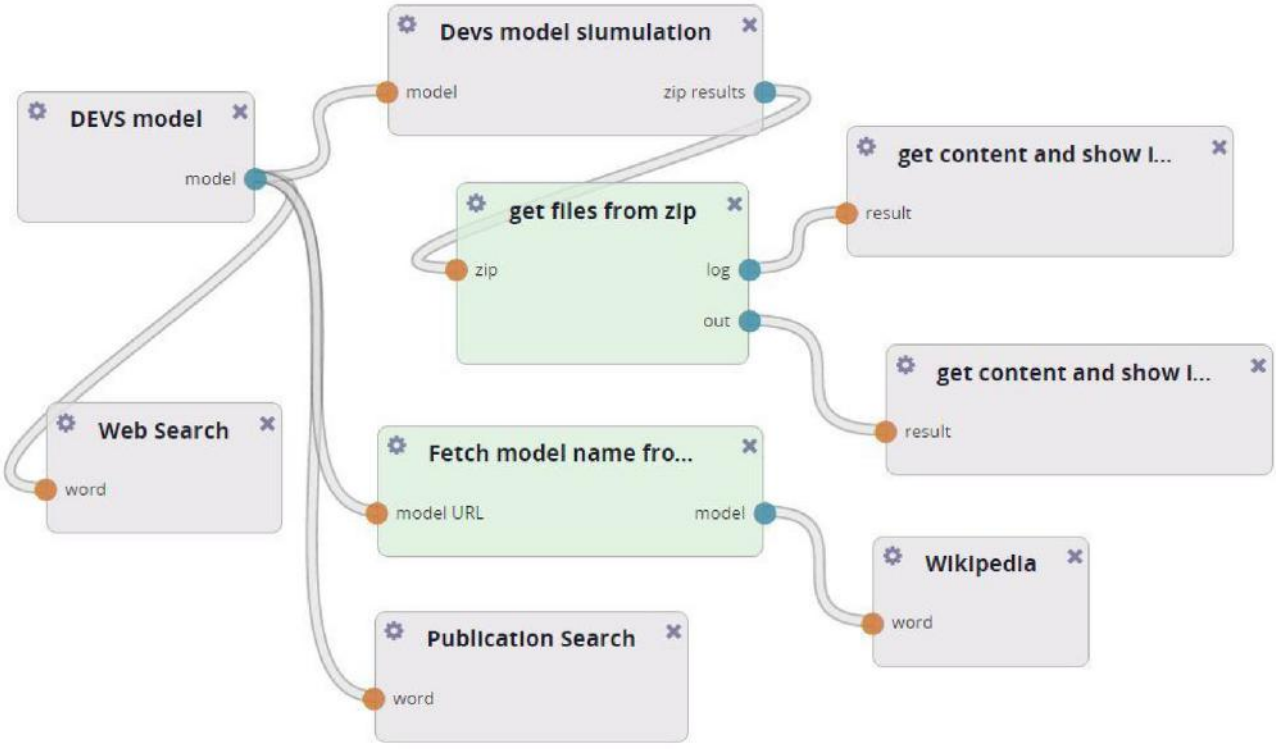


Fig. 19. Boxes wiring for Barber DEVS M&S mashup.

kind of data. For instance, building designers use CAD (Computer-Aided Design) tools to improve the quality of building and urban designs, and BIM (Building Information Modeling) tools to enhance CAD's databases for managing the buildings' data. This mashup was developed by a non-expert user in two week-person time (one undergraduate student without experience in the field, showing the feasibility of the approach for non-expert users). In particular, this mashup includes:

- **Pedestrian modeling**: a Cell-DEVS model to simulate pedestrian evacuation.
- **Pedestrian data collection**: a function for generating initial data files from CAD/BIM file to be used as inputs to the Cell-DEVS model.
- **Pedestrian model simulation**: for executing simulation experiments in the Cloud.
- **Results visualization:** a widget to visualize the simulation result.
- **Flickr pictures:** used as model's state during the visualization.
- **YouTube videos:** videos related to pedestrian M&S.
- **Publications:** existing publications related to pedestrian M&S.

The Box Development Tool is used to develop boxes for the above M&S resources, which in this case included:

- **MSaaS** uses CloudRISE. They contain a *Pedestrian model* (model as a service), a *Pedestrian model simulation* (a simulation experiment for this model), and *Pedestrian data collection* (a function experiment for collecting GIS data as model input).

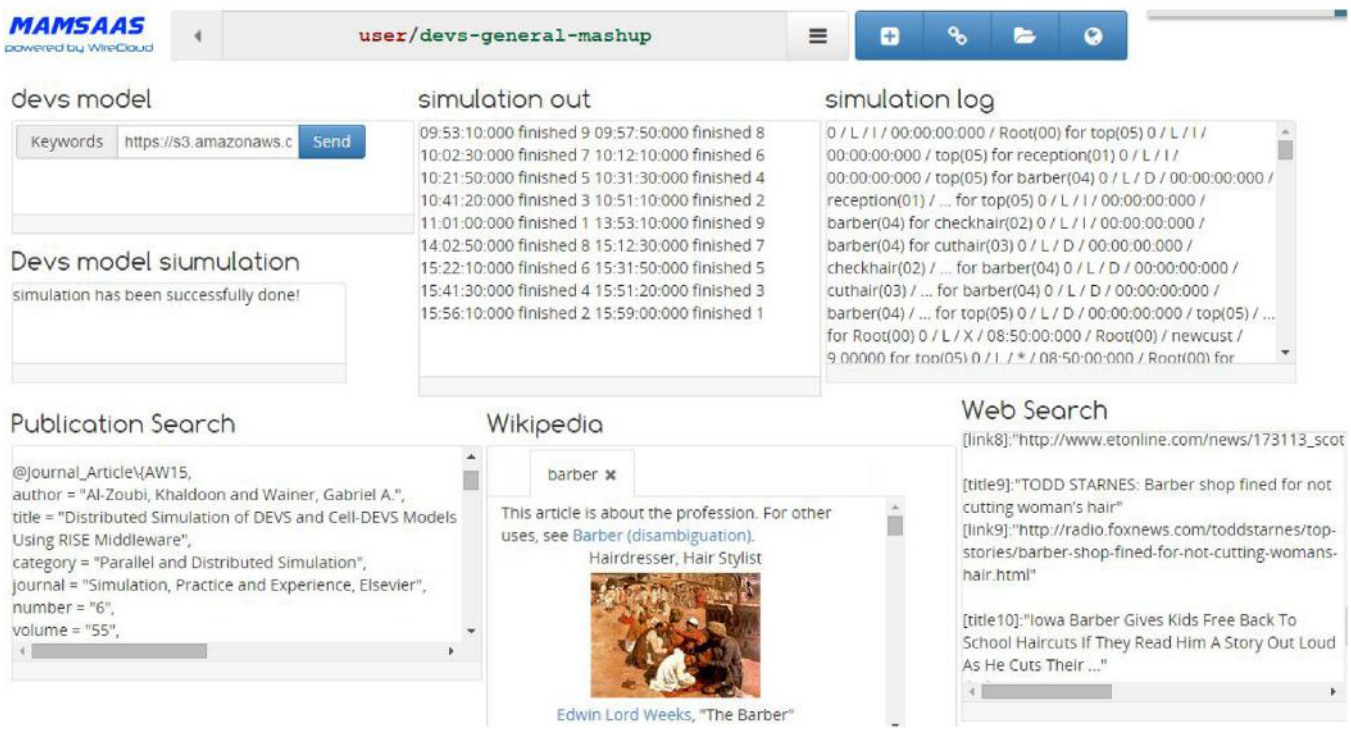


Fig. 20. Executing the Barber M&S mashup application.

- **WebAPI** calls existing open Web APIs. They contain *Flickr keyword search* (a Flickr REST API for searching images of people) and *Publication search* (a REST API to query related publications of given model).
- **Widget** shows input data in web browsers. It contains *Input IFC Box* (a multi-floor building), *Input DSM Box* (a domain-specific model for building layout extraction), *Log Viewer for Pedestrian*, and *YouTube videos* (a widget to show related videos of the given model).
- **Operator Box** handles inconsistencies between boxes. It contains *Log from Zip Box* (extracting the log from the simulation results) and *fetch model name from URL* (extracting the model name from a given URL).

For each Box, its signature was extracted from the corresponding configuration files. The *Pedestrian model* in is a Cell-DEVS model to determine the evacuation time and the occupancy level of a building. The building has multiple floors connected by stairwells. There are exit doors on the first floor. In the case of emergencies, people try to evacuate along the pathway on each floor, moving down towards the exits. Each cell contains the information of direction to the exit. After the model is ready, it is deployed as a Model as a Service in the CloudRISE.

After the boxes are ready, we wire them in Mashup Platform via their inputs and outputs (Fig. 21). First, the links of *Input DSM Box* and *Input IFC Box* wire to the corresponding inputs of Box *Pedestrian data collection,* then its outputs *initialization* and *parameters* wire the inputs *value* and *variable* of Box *Pedestrian model simulation*, respectively. We also wire the outputs of *Pedestrian model simulation* to the inputs of Operator *Log from Zip with pictures*, which later wire to Widget *Log viewer*. In order to get the image links of the cell states, we wire the output port *pedestrianmodel* to Operator *Fetch model name from URL,* then it links Box *Flickr Keyword search,* after which its output wires the input port *pictures* of *Log from Zip with pictures*. In another direction, the *Pedestrian model* wires both Box *Publication Search* and *YouTube videos*.

When the wiring is done, the mashup for this pedestrian M&S mashup is ready; a user can input the URIs of pedestrian model, IFC building file, and DSM model to the input boxes. After that, the mashup will do everything automatically (see Fig. 22). It extracts initial data for the simulation, creates new experiment and executes it for the simulation and visualizes it using images obtained based on the model name.

Thus by entering only the initial files, the user gets a proper visualization with functionalities to play, pause, step and randomly view the simulation in 3D. In addition, we can see the searched Flickr images for cell states, the YouTube videos and the publication lists that relate to pedestrian M&S. Please note that most of the boxes are reusable in other mashups. For example, the log viewer has been developed in a way to handle both 2D and 3D Cell-DEVS models. Similarly, the Flickr box can be re-used by searching new images for the model.

7. Conclusion

Web-based Simulation (WBS, which exposes M&S functions as web services) and Cloud-based Simulation (CBS, which integrates WBS and Cloud Computing) have advanced, but building these environments is still a complicated process as these new technologies are still difficult to develop, deploy and integrate with M&S applications. Here, we presented a new method for dealing with these issues allowing the definition of M&S mashups for fast application development by integrating heterogeneous data and services. We introduced a novel architecture, named the Mashup Architecture with Modeling and Simulation as a Service (MAMS), a layered architecture to deploy and identify M&S mashup component as well as link and execute mashups for quick M&S application development. It has five layers (Cloud, Box, Wiring, Mashup, and Tag Ontology). Here, we focused on the top layers in the hierarchy, showing how to build different applications.

This provides a simplified life cycle to develop, deploy, identify, select, integrate and execute varied M&S resources as services. MAMS uses RESTful WS as the WS framework in its Cloud Layer, tak-

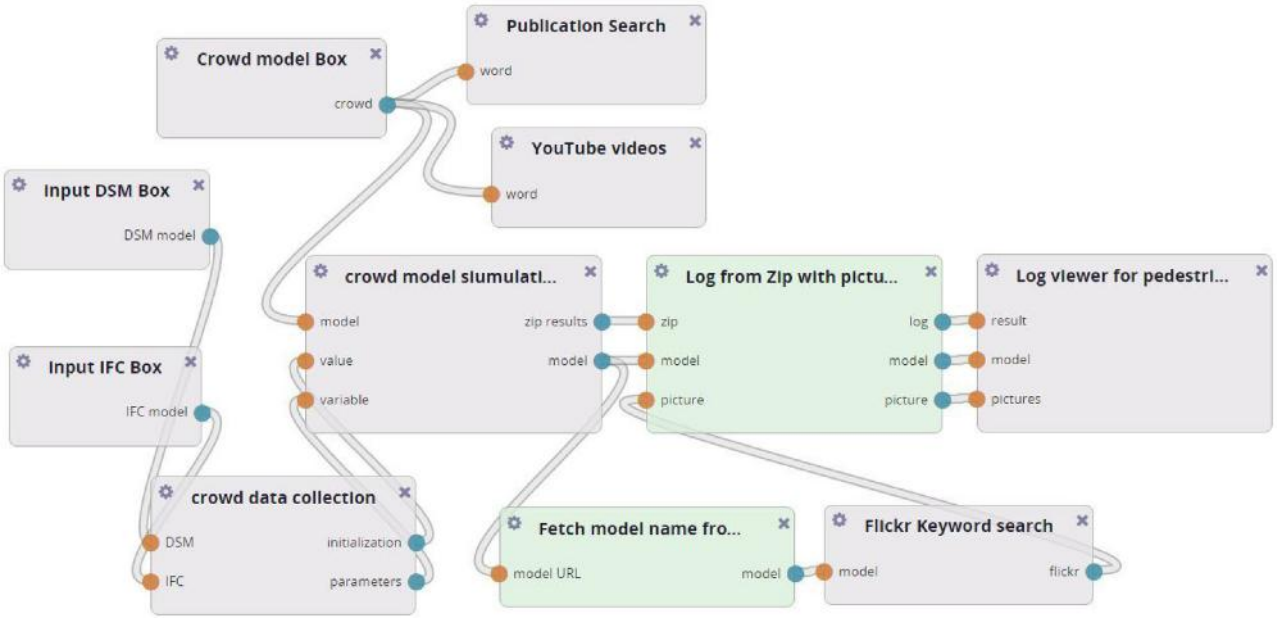


Fig. 21. Wiring boxes of Pedestrian M&S mashup.

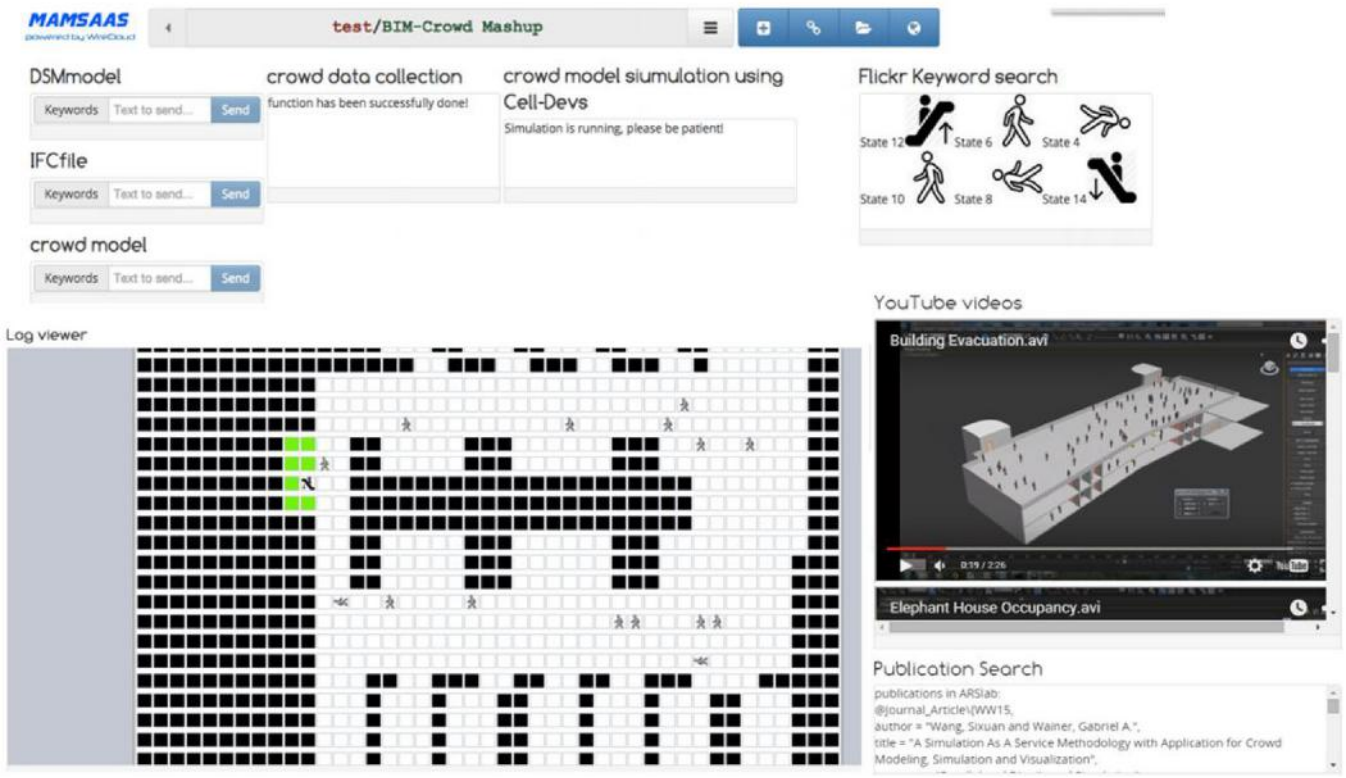


Fig. 22. Pedestrian M&S mashup.

ing full advantages of the Web (every M&S resource is exposed as a unique URL that can be operated on using HTTP methods). MAMS uses the concept of Modeling and Simulation as a Service (MSaaS) in the Cloud, implemented a middleware named CloudRISE, which allows users to develop M&S resources as services in the Cloud. In order to integrate available M&S services and existing WebAPIs, MAMS introduce M&S Mashups. This lightweight M&S application development technique allows organizing M&S resources into MSaaS, WebAPIs, Widgets and Operators, and they are packaged as Boxes (with signature, function and view). A semantic selection approach can mine tags from user-interested resources, learn the tag-tree ontology from the tags, and then select appropriate resources based on their tags and the tag-tree ontology.

The case studies presented show that the proposed architecture can be used to build simulation mashup easily. Box developers can be built by those with basic web development knowledge (like HTML/CSS/JS), and no specific knowledge is required for developing M&S mashups (as users can simply select boxes and wire them),

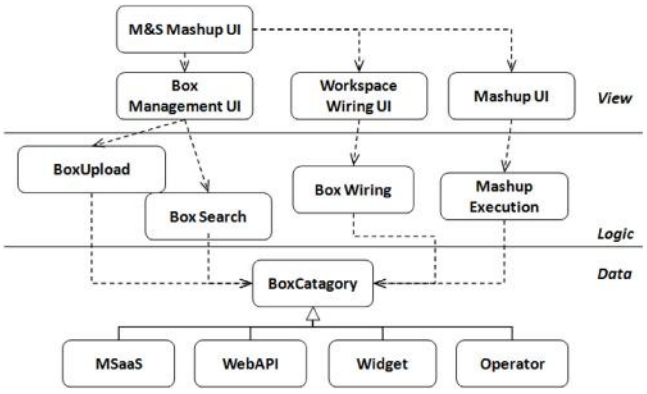


Fig. 23. Class diagram of M&S Mashup Platform (powered by WireCloud).

making the development of these integrated applications easy and feasible. As future work, we plan to flow the changes done to the WireCloud project, allowing other researchers to benefit from this open source project results.

Funding

This research has been partially funded by NSERC.

Appendix A. M&S Mashup Platform using Wirecloud

The M&S Mashup Platform has a wiring editor. This editor allows users to build M&S mashups rapidly. Users drag and drop appropriate boxes into a workspace, and then connect the boxes. The mashup is then ready and users can visualize this mashup at runtime.

The M&S Mashup Platform is an extended version of WireCloud [Zah14]. WireCloud is an open source mashup platform. It supports widget uploading and wiring, user workspace management, and mashup execution. The main difference between WireCloud and our mashup platform is that our platform supports boxes for M&S mashup. WireCloud is a general-purpose mashup platform, but not for M&S mashups (which lacks the support of different boxes with different handling processes). The M&S mashup platform extended WireCloud. It supports boxes for M&S mashup (i.e. MSaaS, WebAPIs, widgets, and operators) (Fig. 23).

The M&S Mashup Platform is implemented in Python using Django. Django is an open-source web application framework. The reason we chose Django is that WireCloud is developed using Django and it has many benefits. Django supports the MVC architectural pattern. Django's primary goal is to ease the creation of complex database-driven websites. Django emphasizes reusability of components and rapid development. Django also provides an optional administrative interface. This interface simplifies the back-end data maintenance. Many well-known sites use Django, including Pinterest, Instagram, and Mozilla.

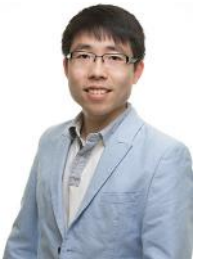
The M&S Mashup Platform follows MVC design patterns, with following classes.

- The *Data* classes extend the *Category* in WireCloud to *BoxCategory*, in which we added two new types, which are *MSaaS* and *WebAPI*. We reuse most features of *widget* and *operator* provided by WireCloud. Each box has three elements: signature, view, and function.
- The *Logic* classes extend the uploading and searching logic of WireCloud to support all types of Boxes. In *BoxUpload*, we changed the package format as zip file, and modified the uploading and parsing logic. In *BoxSearch*, we changed the databases and searching logic. We extended the *BoxWiring* mechanism and *MashupExecution* mechanism in WireCloud to support all types of boxes.
- The *UI* classes change the UI of WireCloud. The overall UI has been modified in *M&S Mashup UI*. For *Box Management UI*, we changed the box uploading and searching pages. For *Workspace UI*, we modified following functions: 1 *Add Box* button in user workspace, 2 *Wire Boxes* button that drags and drops boxes in the wiring editor, 3 *My Boxes* to select from available boxes. For *Mashup UI*, we reuse the mashup execution engine of WireCloud for box execution.

References

[1] S. Aghaee, C. Pautasso, End-user programming for web mashups, in: Current Trends in Web Engineering, 2012, pp. 347–351.
[2] K. Al-Zoubi, G. Wainer, RISE: a general simulation interoperability middleware container, J. Parallel Distrib. Comput. 73 (5) (2013) 580–594.
[3] M. Arroqui, C. Mateos, C. Machado, A. Zunino, RESTful Web Services improve the efficiency of data transfer of a whole-farm simulator accessed by Android smartphones, Comput. Electron. Agric. 87 (2012) 14–18.
[4] S. Balasubramaniam, G.A. Lewis, S. Simanta, D.B. Smith, Situated software: concepts motivation, technology, and the future, IEEE Software 25 (6) (2008) 50–55.
[5] O. Balci, A life cycle for modeling and simulation, Simulation 88 (7) (2012) 870–883.
[6] P. Brebner, Service-oriented performance modeling the MULE enterprise service bus (ESB) loan broker application, Proceedings of 35th Euromicro Conf. on Software Engineering and Advanced Applications (2009) 404–411.
[7] H. Bruneliere, J. Cabot, F. Jouault, Combining model-driven engineering and cloud computing, in: Modeling, Design, and Analysis for the Service Cloud-MDA4ServiceCloud'10: Workshop's 4th Edition, 2010.
[8] J. Byrne, C. Heavey, P.J. Byrne, A review of Web-based Simulation and supporting tools, Simul. Modell. Pract. Theory 18 (3) (2010) 253–276.
[9] E. Cayirci, Modeling and simulation as a cloud service: a survey, Proceedings of the 2013 Winter Simulation Conference (2013) 389–400.
[10] F. Daniel, M. Matera, M. Weiss, Next in mashup development: user-created apps on the web, IT Prof. 5 (2011) 22–29.
[11] R.M. Fujimoto, A.W. Malik, A.J. Park, Parallel and distributed simulation in the Cloud, SCS Magazine 3 (2010) 1–10.
[12] H. Gebhardt, M. Gaedke, F. Daniel, S. Soi, F. Casati, C.A. Iglesias, S. Wilson, From mashups to telco mashups: a survey, IEEE Internet Comput. 3 (2012) 70–76.
[13] C.A. Goble, J. Bhagat, S. Aleksejevs, D. Cruickshank, D. Michaelides, D. Newman, M. Borkum, Bechhofer, S, M. Roos, P. Li, D. De Roure, myExperiment: a repository and social network for the sharing of bioinformatics workflows, Nucleic Acids Res. 38 (July) (2010) W677–W682, Web Server issue.
[14] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M.R. Pocock, P. Li, T. Oinn, Taverna: a tool for building and running workflows of services, Nucleic Acids Res. 34 (July) (2006) W729–W732, Web Server issue.
[16] D. Le Ph+uoc, A. Polleres, C. Morbidoni, M. Hauswirth, G. Tummarello, Rapid semantic web mashup development through semantic web pipes, Proceedings of the 18th World Wide Web Conference (2009) 581–590.
[17] B. Li, X. Chai, B. Hou, T. Li, Y.B. Zhang, H.Y. Yu, X.H. Wang, Networked modeling simulation platform based on concept of Cloud Computing-Cloud simulation platform, J. Syst. Simul. 21 (17) (2009) 5292–5299.
[18] X. Liu, Q. He, X. Qiu, B. Chen, K. Huang, Cloud-based computer simulation, Towards planting existing simulation software into the Cloud, Simul. Modell. Pract. Theory 26 (1) (2012) 135–150.
[19] M.G. Lozano, F. Moradi, R. Ayani, SDR: A Semantic based Distributed Repository for simulation models and resources, First Asia International Conference on Modelling Simulation (2007) 171–176.
[20] R. Madhoun, B. Feng, G. Wainer, On the creation of distributed simulation web-Service-Based distributed CD++, Proceedings of Artificial Intelligence Simulation and Planning (2007) 1–6.
[21] S. Mittal, J.L. Risco-Martín, B.P. Zeigler, DEVS/SOA: a cross-platform framework for net-centric modeling and simulation in DEVS unified process, Simulation 85 (7) (2009) 419–450.
[22] G. Mulligan, D. Gracanin, A comparison of SOAP and REST implementations of a service based interaction independence middleware framework, Proceedings of the 2009 Winter Simulation Conference (WSC) (2009) 1423–1432.
[23] S. Onggo, S. Taylor, A. Tulegenov, The need for Cloud-based Simulation from the perspective of simulation practitioners, Proceedings of the Operational Research Society Simulation Workshop (2014) 103–112.
[24] M. Papazoglou, Web Services: Principles and Technology, Pearson Prentice Hall, 2007.
[25] Radeski, S. Parr, Towards a simulation component model for HLA, Proceedings of the 2002 Fall Simulation Interoperability Workshop (2002).
[26] L. Richardson, S. Ruby, RESTful Web Services, O'Reilly Media, 2007.
[27] J. Ribault, G. Wainer, Using workflows of web services to manage simulation studies into the cloud, in: Proceedings of SCS/ACM Symposium on Theory of Modeling and Simulation, TMS/DEVS'12, Orlando, FL, 2012.

[28] S. Robinson, R.E. Nance, R.J. Paul, M. Pidd, S.J. Taylor, Simulation model reuse: definitions, benefits and obstacles, Simul. Modell. Pract. Theory 12 (7) (2004) 479–494.
[29] R.V. Rocha, R.B. Araujo, M.R. Campos, A. Boukerche, Understanding and building interoperable, integrable and composable distributed training simulations, 2010 IEEE/ACM 14th International Symposium on Distributed Simulation and Real Time Applications (2010) 121–128.
[30] P. Siriwardena, Advanced API Security, Springer Press, 2014.
[31] A. Skoogh, T. Perera, B. Johansson, Input data management in simulation–Industrial practices and future trends, Simul. Modell. Pract. Theory 29 (2012) 181–192.
[32] L. Sliman, B. Charroux, Y. Stroppa, A new collaborative and cloud based simulation as a service platform: towards a multidisciplinary research simulation, 15th International Conference on Computer Modelling and Simulation (2013) 611–616.
[33] S.J.E. Taylor, A. Khan, K. Morse, A. Tolk, L. Yilmaz, J. Zander, CAMBIARLO POR EL MIO grand challenges on the theory of modeling and simulation, Proceedings of the Symposium on Theory of Modeling and Simulation (2013) 34.
[34] S.J.E. Taylor, T. Kiss, G. Terstyanszky, P. Kacsuk, N. Fantini, Cloud computing for simulation in manufacturing and engineering: introducing the CloudSME simulation platform, Proceedings of the 2014 Annual Simulation Symposium (2014), pp. 12.
[35] Tolk, S. Mittal, A necessary paradigm change to enable composable Cloud-based M&S services, 2014 Winter Simulation Conference (2014) 356–366.
[36] T.D. Trinh, P. Wetz, B.L. Do, A. Anjomshoaa, E. Kiesling, A.M. Tjoa, Open linked widgets mashup platform, 2014 ESWC (2014).
[37] W.T. Tsai, W. Li, X. Bai, J. Elston, P4-simsaas: policy specification for multi-tendency simulation software-as-a-service model, Proceedings of the 2011 Winter Simulation Conference (2011) 3067–3081.
[38] K. Wagh, R. Thool, A comparative study of soap vs rest web services provisioning techniques for mobile host, J. Inf. Eng. Appl. 2 (5) (2012) 12–16.
[39] G. Wainer, Applying cell-DEVS methodology for modeling the environment, Simulation: Trans. Soc. Model. Simul. Int. 82 (10) (2006) 635–660.
[40] G. Wainer, Discrete-Event Modeling and Simulation: A Practitioner's Approach, CRC Press, 2009, 2017.
[41] G. Wainer, R. Madhoun, K. Al-Zoubi, Distributed simulation of DEVS and Cell-DEVS models in CD++ using Web-Services, Simul. Modell. Pract. Theory 16 (9) (2008) 1266–1292.
[42] S. Wang, G. Wainer, Semantic mashup for simulation as a service with tag mining and ontology learning, Proceedings of the Symposium on Theory of Modeling and Simulation (2014), pp. 25.
[43] S. Wang, G. Wainer, A simulation as a service methodology with application for crowd modeling, simulation and visualization, Simulation: Trans. Soc. Model. Simul. 91 (1) (2015) 71–95.
[44] S. Wang, G. Wainer, Semantic selection for model composition using SAMSaaS, Proceedings of the Symposium on Theory of Modeling and Simulation, Society for Computer Simulation International (2015) 25–32.
[45] S. Wang, G. Wainer, A mashup architecture with modeling and simulation as a service, International Conference on Web Information System Engineering (2015) 247–261.
[46] G. Wang, S. Yang, Y. Han, Mashroom end-user mashup programming using nested tables, Proceedings of the 18th World Wide Web Conference (2009) 861–870.
[47] T. Zahariadis, A. Papadakis, F. Alvarez, J. Gonzalez, F. Lopez, F. Facca, Y. Al-Hazmi, FIWARE lab: managing resources and services in a cloud federation supporting future internet applications, in: Utility and Cloud Computing, 2014, pp. 792–799.
[48] B.P. Zeigler, H. Praehofer, T.G. Kim, Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems, Academic press, 2000.
[49] B.P. Zeigler, L. Zhang, Service-oriented model engineering and simulation for system of systems engineering, in: Concepts and Methodologies for Modeling and Simulation, 2015, pp. 19–44.



Sixuan Wang received his PhD degree in Electrical and Computer Engineering with the Department of Systems and Computer Engineering at Carleton University. He received double master's degrees from Harbin Institute of Technology, China, and the University of Bordeaux 1, France. His academic experience spreads over a wide range of areas, such as M&S, software engineering, distributed systems, big data and cloud computing. He has participated in many scientific and industrial projects with different organizations, including NSERC, SINTEF, SFU, Autodesk, SageTea, Honeywell and Qlik. He has over 10 publications and is a frequent reviewer of scientific papers. His current research interests are M&S as a service, cloud computing, micro-services, mashup and data integration.

Gabriel A Wainer (SMSCS, SMIEEE) is a full professor at the Department of Systems and Computer Engineering at Carleton University. He is the author of four books and over 320 research articles; he has edited four other books and helped organize over 170 conferences, including being one of the founders of TMS/DEVS, SIMUTools and SimAUD. He is the principal investigator of different research projects. He is the vice-president of conferences. He is special issues editor of Simulation. He is the head of the Advanced Real-Time Simulation Lab, located at Carleton University's Centre for advanced Simulation and Visualization (V-Sim). He has been the recipient of various awards, including the IBM Eclipse Innovation Award, SCS Leadership Award and various Best Paper awards. His current research interests are related to modeling methodologies and tools, parallel/distributed simulation and real-time systems.