A Mashup Architecture with Modeling and Simulation as a Service

Sixuan Wang, Gabriel Wainer

Dept. of Systems and Computer Engineering, Carleton University
1125 Colonel By Dr. Ottawa, ON K1S5B6, CANADA
{swang, gwainer}@sce.carleton.ca

Abstract. Web services have been used in Modeling and Simulation (M&S) for many years, but it is still hard to develop complex M&S application by using heterogeneous data and varied services. Here we show how to simplify the M&S application development by using mashup technologies. We introduce a novel architecture, the Mashup Architecture with Modeling and Simulation as a Service (MAMSaaS), in order to deploy varied mashup components (e.g. Modeling and Simulation as a Service, existing Web APIs, widgets and operators) and to create M&S mashups for quick application development. We present various tools built to support the architecture and a case study using the mashup architecture.

Keywords: Modeling and Simulation as a Service · Cloud-based Simulation · Web Data Integration and Mashups · Service Composition · Web API.

1 Introduction

Current Modeling and Simulation (M&S) systems have become more and more complex, involving people different with M&S expertise, and varied M&S resources. In order to simplify the development process of M&S systems, the M&S community has used Web-based Simulation (WBS), Cloud-based Simulation (CBS) and Web Services (WS) technologies for around 20 years. WBS focused on running experiments using the Web and later exposing M&S functions as web services [1]. This method has been successful, and a large number of M&S WSs exist, including SOAP-based WS [2] and RESTful WS [3]. CBS integrates WBS and Cloud Computing. It uses Cloud Computing technologies to reduce costs and make easier to develop M&S systems by exposing M&S resources as Modeling and Simulation as a Services (MSaaS).

Nevertheless, the growth of M&S related WSs and open Web APIs makes it difficult to integrate them for complex applications. In particular, experts from different

domains cannot easily use existing M&S related resources for fast application development. With this in mind, we aim to improve the development process of Web information systems for M&S that involve heterogeneous data and services. This process should be such that one could reuse and integrate existing services and resources related to M&S easily. People from different domains should be able to use this process for sharing and combining their works quickly.

In order to do so, we present an architecture based on mashup technologies in the Web 2.0 [4]. Mashups have been widely used in different domains; however, these technologies have never been used to simplify the process of building M&S applications. Our objective is thus to develop M&S mashup applications to integrate different M&S related WSs and resources. Our novel architecture, named Mashup Architecture with Modeling and Simulation as a Service (MAMSaaS), is a layered architecture to deploy and identify M&S mashup components as well as link and execute mashups for quick M&S application development.

MAMSaaS supports universal identification and development for mashup component (named *Boxes*), which consist of varied M&S resources (MSaaSs, WebAPIs, widgets and operators). In order to link and execute these components, MAMSaaS supports component linking, development and search boxes, then build M&S environments by wiring boxes, and then execute and visualize M&S at run-time.

Following, we will discuss the architecture and the tools we implemented: a Box Development Tool and the MAMSaaS Mashup Platform. In addition, we also present a prototype application integrating a Geographical Information System (GIS), a distributed M&S tool, and visualization services, built as a mashup application.

2 Background

M&S is being applied to almost every aspect of life, and developing of M&S applications has become more and more complex [6]. Varied resources are involved in the development process, such as source systems, models, simulators, experimental frameworks and experiments [7], as well as supported data (e.g. text, file, database) and functions (e.g. data collection, result analysis, visualization) [8].

In order to simplify the application development process, web services have been used in M&S for around 20 years. The basic idea is to expose M&S resources on the Web as services. Web services ease the sharing of M&S resources: for instance, the simulators are located remotely on a server, without worrying about the simulation environment setup and software dependencies. Web services improve data accessibility, interoperability and user experience [9].

In general, web services in M&S include Web-Based Simulation (WBS, which exposes M&S functions as web services) and Cloud-Based Simulation (CBS, which integrates WBS and Cloud Computing). In WBS, the functions of simulators and their simulation environment are exposed as web services [1]. 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. In recent years, numerous WSs in WBS have been developed, which can be categorized

into two main frameworks: SOAP-based (e.g. DDSOS [10], SOPM [11], and SASF [12]) and RESTful-based (e.g. RISE [3], RESTful MMVE [13], and RESTful AIS [14]). Numerous simulators (i.e., DEVS/SOA [2] and RISE [3]) implement the DEVS M&S formalism [7] over a WS. RISE is the first and only RESTful distributed simulator to support distributed simulation that supports DEVS and other model formalisms, languages and engines.

In CBS, both web services and Cloud computing are used in M&S. CBS is derived from WBS, using Cloud Computing to manage varied M&S resources and build different simulation environments [15]. The use of web services in CBS has received the name of Modeling and Simulation as a Service (MSaaS). MSaaS is a special form of Software-as-a-Service (SaaS), as it hides the underlying infrastructure, platform and software details from the users. The use of CBS and MSaaS is still in a preliminary stage [16] and little effort has been done to integrate with other services [17], which is an issue because there are many open Web APIs that could be useful for M&S applications (e.g. weather forecast, GIS information, and big data for simulation inputs). They could improve user experience and make richer applications [18].

According to more than 11,000 APIs registered by ProgrammableWeb, REST WSs take 73% while SOAP-based APIs take 27% [19]. WADL is a popular language to describe REST web services, and many IT companies describe their REST APIs on the 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. However, there has been no research showing how to integrate these useful Web APIs in the development process of M&S applications. Mashup technologies in Web 2.0 can be used to solve this integration issue and simplify the M&S application development.

Mashups integrate different services from the web, using content from more than one existing source to create a new value-adding application [4]. Mashups integrate heterogeneous data, application logic (exposed as services in general), and UI components (e.g. widgets) [5]. A large number of mashup techniques and tools have been developed in both industrial development and academic research [5]. Many industrial 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 (DIY) guidance to meet user requirements [20]. In addition, many academic efforts focus on mashups. Many of them use End-User Programming (EUP), focusing on the composition and integration of web sources for new purposes. Mashroom [21] uses nested relational models and provides mashup operations like merge and filter over tables. In [22], the authors use native language programming in mashup components, linking different logic together.

The fundamental element of current mashup technologies is the *widget*, a small processing unit for performing single purpose task such as fetching, parsing, formatting and visualizing data [23]. For instances, DERI Pipes [24] enables users to build widgets to process data from different sources (e.g. RDF, SPARQL, XML, HTML). In [23], the authors proposed an open mashup platform with linked widgets created freely by users that can be discovered and combined easily. WireCloud [25] is an

open source mashup platform provided by the FI-WARE project, which can implement widgets in JavaScript (JS) and HTML5 and build mashups by "wiring" widgets.

However, no one has ever tried to develop such widgets and mashup applications in M&S (as well as integrating available WBS, CBS and Web APIs). Many of the current mashup techniques and tools cannot work directly for M&S because: 1) they are domain specific (i.e., they are useful only for single or limited problems in specific domains); 2) they have been discontinued; 3) they are limited (the widgets do not support MSaaS and different kinds of Web APIs like SOAP and REST). Based on this, we investigated a new mashup method focused on the process of developing M&S.

3 The MAMSaaS Architecture

Based on the considerations in Section 2, we defined a novel mashup architecture to simplify the process of developing M&S applications, named Mashup Architecture with Modeling and Simulation as a Service (MAMSaaS). Shown in Fig. 1, MAMSaaS is a layered methodology and architecture to create and run M&S mashups.

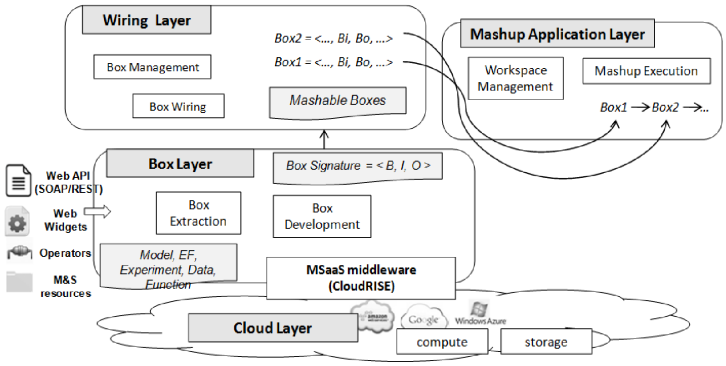

Fig. 1. MAMSaaS Architecture

The proposed MAMSaaS architecture has four layers, as follows:

- **Cloud:** it is responsible for supporting Cloud infrastructure and deploying MSaaS. The Cloud infrastructure includes Cloud *compute* units (for building and executing simulation experiments) and Cloud *storage* units (for sharing M&S resources). In addition, this layer is also responsible for deploying user-provided M&S resources as MSaaS in the Cloud by using the *MSaaS middleware* on-demand.
- **Box:** it is responsible for developing mashup components (termed *Boxes*). Boxes can have different categories (e.g. MSaaS from the Cloud Layer, existing open APIs, widgets, operators). Each box is identified by a uniform box signature, and has its own function for handling input messages and has its own visual form.

- **Wiring:** it is responsible for connecting boxes into a mashup. Boxes can be linked with each other by their inputs/outputs that are identified in their box signatures. A same box can be reused and re-wired in different mashups for new purposes.
- **Mashup Application:** it is responsible to select and wire boxes in workspaces, and run applications. They can add box, wires and visualize the results at runtime.

Here we will discuss the top three layers: Box, Wiring, and Mashup Application layers, and discuss how to develop boxes and build M&S mashups. For the details about the Cloud layer, the reader can refer to [26], in which we introduced the CloudRISE middleware, which implements the concept of MSaaS (exposing all kinds of M&S resources as services). CloudRISE uses a resource-oriented design via RESTful WSs in which M&S resources are identified through URIs in the Cloud.

4 M&S Mashup Methodology

Box, wiring, and M&S mashups make up the complete set of an M&S mashup application. In this section, we will discuss details of these three concepts.

4.1 Box – developing Mashup components

Boxes represent mashup components used for M&S. They are mini applications for M&S-related scenarios (e.g. fetching a model, reproducing an experiment, visualizing results, etc). Boxes receive heterogeneous data from varied services. A Box modularizes specific functions and sends data to others; they can be shared or published on the Web. There are four basic types: *MSaaS, WebAPI, Widget*, and *Operator* Boxes.

- **MSaaS Box**: it uses MSaaS services from CloudRISE to handle input data, and then outputs results. CloudRISE works as a repository interface to expose M&S resources as MSaaS. There are six main types: simulations, supporting functions, models, supporting data, semantic data and instances. The ones related to the M&S mashup are simulations, functions, and models. Users can manage resources and control the lifecycle of the execution of simulations and functions by using the HTTP methods GET/PUT/POST/DELETE to corresponding URIs. MSaaS Boxes simplify the execution of experiments by combining several MSaaS into one box. For example, an MSaaS simulation box can create a new experiment configuration file, create a new experiment using that file, start the simulation and check its status; when it finishes, it sends the simulation results to the output.
- **WebAPI Box**: it calls existing open Web APIs. It exposes the function following a WS principle (RESTful/SOAP-based). RESTful Web APIs are usually described in WADLs or HTML pages, while SOAP-based Web API is described in WSDLs.
- **Widget Box**: it is a lightweight web application that shows the data on web browsers. They provide a visual representation for particular data. They can be reused for web development or other mashup platforms. For example, http://www.100widgets.com provides different widgets, supporting varied type of data, e.g. forms, diagrams, tables, maps, photos and videos.

- **Operator Box**: it takes input data from other boxes and it generates the output based on a customized process. The reason why we need operators is to address the inconsistencies between boxes (i.e. boxes with similar ports that cannot connect directly). Operators can be viewed as a converter between boxes. It can be a filter, aggregator, splitter, or adapter. For example, if one port of a box is *full name*, and one port in another box is *given name*, an operator can be a splitter that extracts *given name* from *full name*, so these boxes can be connected.

Though boxes have different types, they are managed in a similar way. Each box is packaged in a separated archive file, so it can be developed, downloaded and installed on different servers. Each box package has three parts, as follows:

- Box Signature: To manage the varied boxes, we provide a uniform structure,

Box Signature (B) = < B_x, I, O> **(Definition 1)**

B $_{\boldsymbol{x}}$ **=** $<B_n,B_t,B_{d,},B_s,B_{a,}B_p,B_m>$ *is the general information of the box,*
I *= {p} is a set of input ports of the box,*
O *= {p} is a set of output ports of the box,*
p **=** $< p_n, p_t, p_d >$ *includes port name, type, and description*.

The Box Signature identifies each box with its basic information (B_x), input ports (I) and output ports (O). The basic information includes its name B_n, type B_t (e.g. MSaaS, WebAPI, Widget, Operator), description B_d, subtype B_s (e.g. simulation MSaaS, RESTful WebAPI), author B_a, path B_p (the URI of a related WS), and method B_m (the method name of a related WS). A box can have multiple input and output ports, which are used to connect the boxes in a mashup. Each input or output port include a port name, type (the message type in the port) and description (a text that describes the port). For instance, given a WebAPI RESTful WS to forecast the weather, its name is *WeatherForecast*, type is *WebAPI*, description is *return weather forecast information*, subtype is *SOAP*, author is *JohnDoe*, path is *http://www.webservicex.net/weatherforcat*, and method is *GetWeather*. This WebAPI Box has one input port (with *CityName* as its name, *xsd:string* as its type and *city name* as its description), and one output port (with *weather* as its name, *xsd:string* as its type and *weather forecast in 5 days* as its description).

Box Signatures can be described in XML files that can be provided by users or constructed automatically from existing sources.

- Box Function: Each box has a function to response the input events. Boxes are event-driven. When an input event comes, it triggers a function in the Box; then, it sends data through output ports. Different types of boxes have different functions. For the MSaaS Box, it combines multiple MSaaSs related to a same experiment into a single box. For example, for the box of executing a new simulation experiment, box function will: 1) construct a new experiment.xml; 2) create a new experiment; 3) start the simulation; 4) check the result; 5) get result and send it to the output ports. For the WebAPI Box, the function executes as defined in the Web

API (e.g. SOAP WS, RESTful WS). For the Widget Box, the function tells how to handle the input data for visualizing in web browsers. For the Operator Box, the function is the action to be executed (e.g. splitting, combing, and data conversion).

- Box View: Each box can also have a view in web browsers. Boxes can have HTML/CSS files for visualization purposes. For boxes of MSaaS and WebAPIs, their views could be either their signatures or the execution status. For Widget Boxes, they can reuse existing HTML/CSS files in existing widgets. Users can also customize these files to change the view how the data will show.

4.2 Box Wiring – linking boxes

A key feature of the boxes is that they can be connected to each other, which is called *Box Wiring*. It is for composing different boxes through inputs and outputs.

Box Wiring (W) = $<\{B\}, \{B_{x\text{-}I}, B_{y\text{-}O}\}>$ **(Definition 2)**

B *= {B_x, B_y...} is a set of boxes,*
{$B_{x\text{-}I}$, $B_{y\text{-}O}$} *is a set of connections between boxes*

A Box Wiring is a combination of boxes and connections among them. Each Wiring (*W*) contains a set of boxes *{B_x, B_y...}* and connections *{$B_{x.I}$, $B_{y.O}$}*. For instance, $B_{x.I}$, $B_{y.O}$ means the output port *O* of Box B_y can be linked to the input port *I* of Box B_x. Boxes notify about their changes via events on their output ports; other boxes can consume these events via input ports. By wiring boxes, users can reuse them in multiple M&S scenarios without understanding the internal details.

4.3 M&S Mashup – building mashup applications

The boxes and wiring mechanism can be used to build *M&S mashups*. They are used to build a composite M&S application by selecting and wiring boxes. It is based on data flow and event-based mechanism among the boxes using a visual representation.

Mashup (M) = < {B}, {W}, U > **(Definition 3)**

B *= {B_x, B_y...} is a set of boxes,*
W *is a set of Box Wirings,*
U *is a user workspace for this mashup.*

An M&S Mashup consists of a set of Boxes (*B*), Box Wirings (*W*), and a User Workspace (*U*). Users can have different workspaces. In any workspace, users select boxes and wire boxes through their input/output ports. After that, the mashup application is ready. Users can run and visualize it.

Fig. 2 shows an example of an M&S Mashup. It is made up of six boxes (*B1* to *B6*), which interoperate with each other by exchanging data. Consider we have MSaaS Boxes *B1* and *B2*, Web API Boxes *B3* and *B4*, Operator Box *B5*, and Widget

Box *B6*. Users can build a mashup application by wiring these boxes as shown in the right part of Fig. 2. At run-time, the data generated in B1 will pass to B3 and trigger B3's function, and then B3 will output its data to B2. Similar actions happen in other boxes. Users can see the mashup as defined in each Box View of boxes on web browsers.

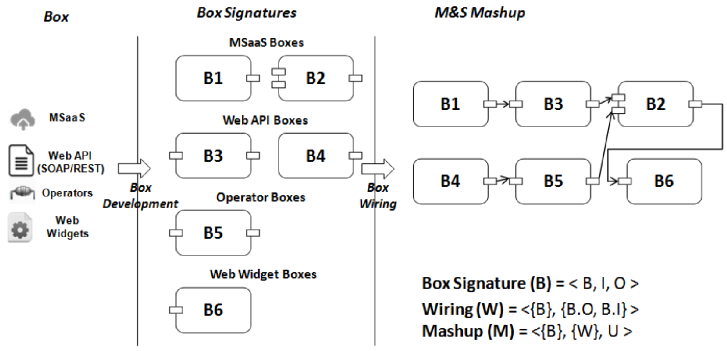

Fig. 2. Box/Wiring/Mashup example

5 MAMSaaS Implementation

We have developed different tools supporting MAMSaaS and the M&S Mashup methodology. Here we discuss the Box Development Tool (used to extract and develop boxes) and the Mashup Platform (used to wire boxes and run M&S mashups).

5.1 Box Development Tool

The Box Development Tool is used to develop boxes rapidly. It can load and save Box Signatures in XML for the different kinds of Boxes. It can also extract Box Signatures from existing files (e.g. MSaaS Experiment Frameworks, WSDL for SOAP-based WebAPI, and WADL for REST-based WebAPI). In addition, it can generate the configuration XML file that is used in the box package. This tool can also suggest users with similar existing boxes.

The Box Development Tool was developed in Java using SWT (Standard Widget Toolkit), which is a graphical widget toolkit. The Box Development Tool follows the MVC design pattern. Fig. 3 shows its class diagram, which consists of three groups:

- The *Data* classes manage the data contained in the *BoxSignatures*. *BoxSignatures* keeps all the signatures in a list. *Signature* keeps the information of a box. *Operation* contains the information of each port. Each *operation* has one *parameter* or *ComplexParameter*. *Parameter* ports have basic information (name/type/description) and *ComplexParameter* uses complex data types (e.g. user-defined XSD).

- The *Logic* classes are used to define the logic of the boxes. They use *Signature* as a bridge. They can extract information from other files to construct a *Signature*, and convert it to other files. *SimulationFramewrokXMLLoader* and *FunctionFrameworkXMLLoader* load Experimental Frameworks from CloudRISE, extracting information from it and saving it as a *Signature* in Data. Similarly, *WADLReader* and *WSDLReader* load and parse the description file of Web APIs. *SignatureXMLHandler* loads and saves the box signature XML files. *Signature 2ConfigurationXMLConverter* converts a *Signature* into a box configuration file.
- The View classes are used to build User Interfaces (UI). It supports user-friendly UI to control the process of generating Boxes. The default UI is *MainShell*. It has two menus: *BoxSignatureUI* for extracting and loading a *Signature*, and *BoxDevelopmentUI* for saving a *Signature* and developing box packages.

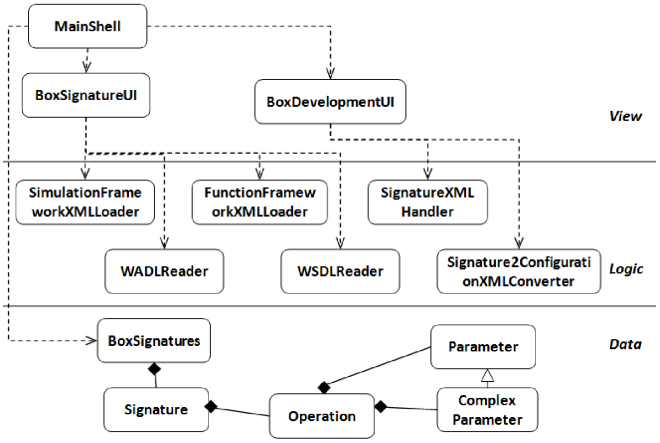

Fig. 3. Class Diagram of Box Development Tool

As discussed before, each box is packaged in an archive file. This file contains a Box Signature, a Box Function and a Box View. In our tool, Boxes are developed using current web technologies (XML, JS, and HTML/CSS). This archive file is an extended version of the widget package used in WireCloud [25], which will be discussed in the next section. The Box Signature is a XML configuration file. 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 the configuration XML file. For the JS and HTML/CSS files, it can suggest users with similar boxes, in order to reuse them. In particular, the JS function triggers functions for input events by reusing the WireCloud's API *MashupPlatform.wiring. registerCallback (inputName, callback);* and when the function finishes, it outputs message by reusing the WireCloud's API *MashupPlatform.wiring.pushEvent(outputName , data).* Inside the JS function, it can execute SOAP or RESTful WS.

Fig. 4 shows an UI example of the Box Development Tool after extracting the Box Signature from an Experimental Framework in the cloud. Users can select the existing MSaaS file, load it, and then the tool parses the file and extracts the information needed for its Box Signature. Later, users can modify this signature, and save it in the format used by the M&S Mashup Platform.

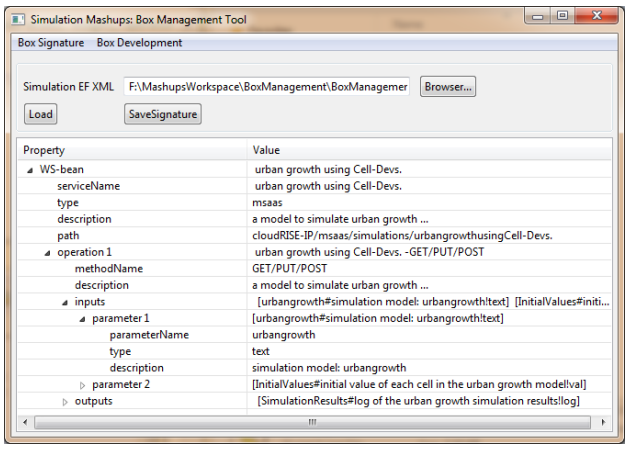

Fig. 4. Extracting Box Signature from MSaaS file.

5.2 M&S Mashup Platform

The M&S Mashup Platform has a wiring editor that allows users rapidly building M&S mashups. Users choose a type of box, drag and drop appropriate boxes into a workspace, and then connect the output of a box to the input of another one. After that, the mashup is ready and users can visualize the mashup at run-time.

The M&S Mashup Platform is an extended version of WireCloud [25], an open source mashup platform. It supports widgets uploading and wiring, user workspace management, and mashup execution. However, WireCloud does not support different type of Boxes (in particular the MSaaS Box and WebAPI Box). WireCloud is a general-purpose mashup platform, but not for M&S mashups (which should manage different boxes with different handling processes). The M&S mashup platform we developed extended WireCloud to support boxes for M&S mashup, which are MSaaS, WebAPIs, widgets, and operators. Fig. 5 shows its class diagram, it consists of three groups of 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 archive file, and modified the uploading and parsing logic. In *BoxSearch*, we changed the databases of resources 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 functions with *Add Box* button to add boxes into user workspace, *Wire Box* button that drags and drops boxes in the wiring editor, *My Box* to select from available boxes. For *Mashup UI*, we reuse the mashup executing engine of WireCloud for box execution.

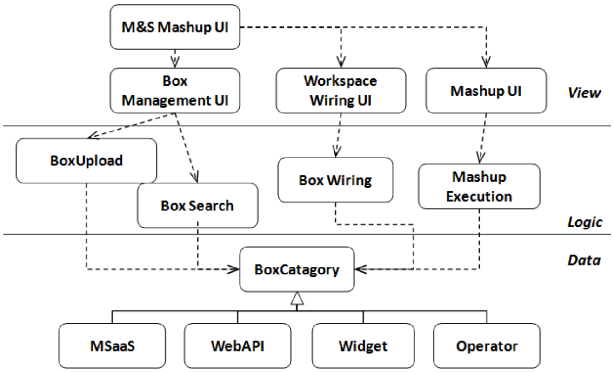

Fig. 5. Class diagram of M&S Mashup Platform (powered by WireCloud)

6 Case Study: a Land Use Modeling application

In this section, we show how to use MAMSaaS and our tools to build and visualize an M&S mashup application from scratch. The case study focuses on building a Land Use M&S mashup using GIS and other related M&S resources. GIS allows managing, analyzing, and displaying geographically referenced information, and it has been studied for several years in M&S. However, developing M&S applications using GIS is still a complex process [27]. GIS M&S requires many M&S related resources including experts from different domains.

Changes for Land Use have drawn much attention in urban planning, engineering, urban economics, and related fields. Land use can also affect the development of transportation, population, and land distribution. Our mashup application includes:

- **Environmental modeling**: we built a Cell-DEVS environmental model to simulate the land use scenarios.
- **Data collection**: we need a function for generating initial data files from GIS to be used as inputs to the Cell-DEVS model.

- **Cloud-based simulation**: we execute simulation experiments in the Cloud.
- **Results analysis**: it is a function to analyze simulation results, e.g. parsing, converting, statistical analysis.
- **Visualize results:** it is a widget in web browsers to visualize the simulation result in a vivid way.
- **WebAPIs:** they are existing web APIs useful for the GIS M&S. For instances, to better predict the landuse tread, web API to forecast weather for the studied GIS area is needed; to know the zip code information, a web API to search zip code based on geographic information is helpful.

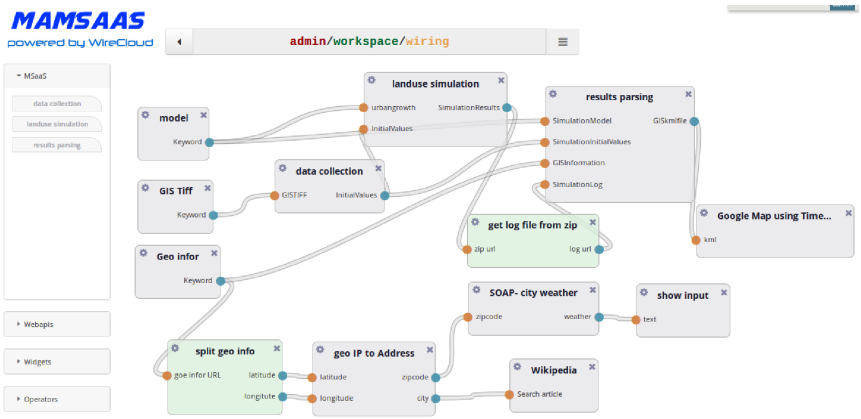

Fig. 6. Boxes wiring for GIS M&S mashup.

The Box Development Tool can help users develop boxes for the above M&S resources. The developed boxes for this case study are as follows:

- **MSaaS Boxes** use CloudRISE. They contain *Landuse model simulation* (a simulation experiment for the landuse model), *GIS_data_collection* (a function experiment for collecting data), and *GIS_KML_analysis* (a function experiment to parse simulation results to KML file). For each MSaaS Box, its signature is extracted from the corresponding configuration files; its function combines multiple MSaaS in a same experiment framework; and its view shows its execution status.
- **WebAPI Boxes** call existing open Web APIs. They contain *City_weather* (a SOAP WS to forecast weather) and *GeoIP_to_address* (a RESTful WS to get an address from a GIS). For each WebAPI Box, its signature is extracted from the existing WS description file or HTML files; the function calls the Web API through HTTP request/response; and the view shows its output message.
- **Widget Boxes** show input data in web browsers. They contain *Input_box* (allowing users to input messages), *Input_show* (showing input messages), *KML_viewer* (viewing KML files in Google Map), and *Wikipedia* (getting Wikipedia informa-

tion). For each Widget Box, its signature is provided by users; the function analyzes input data; and the view visualizes input data.

- **Operator Boxes** handle inconsistencies between boxes. They contain *Zip_to_log* (extracting Log files from an archive), and *Geo_spliter* (splitting Geo information into coordinates). For each Operator Box, its signature is provided by users; the function converts the input data and outputs it; and there is no view for operators.

After the boxes are developed, user can upload them into the M&S Mashup Platform. Now it is time to wire these boxes into a mashup. Fig. 6 shows the box-wiring page in the M&S Mashup Platform. In the wiring editor, users can drag and drop boxes from different types, wire the boxes with their input and output ports. There are three user inputs: *model* (the land use model in CloudRISE); *GIS Tiff* (the user-selected area in a GIS dataset) and *Geo Info* (the global geographical references of GIS dataset). We can wire the output of *GIS Tiff* as input of *GIS Data collection*, to send *GIS Tiff* and extract an initialization file for the *Land Use Simulation*. Similarly, we can wire *model* and *GIS data collection* to *Land Use Simulation* to receive inputs of the model and initial files. Then, it can run the simulation. After that, we can wire the *Land Use Simulation* with the Operator Box *get log file from zip* to extract the log file and then wire this to *Results Parsing*, so a KML file can be generated, which wires to the Widget Box *Google Map* for visualizing purposes. In another path, the *Geo Info* wires to Operator Box *Split Geo Info* to get the coordinates of the area under study, then the coordinates link to the WebAPI Box *get IP to Address* to get *zipcode* and *city address*. After that, the *zipcode* wires to WebAPI Box *City weather* for getting the weather forecast (shown in the Widget Box *ShowInput*); and *city address* wires to Widget Box *Wikipedia* to get wiki information.

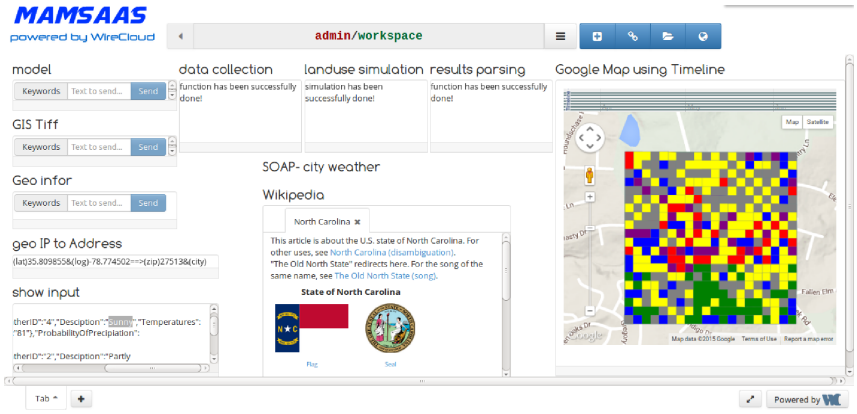

Fig. 7. Executing the GIS M&S mashup application

After the boxes have been wired, a new M&S mashup application is ready. Users can go back to the workspace and execute it. After setting three inputs (i.e. *Model URL; GIS Tiff URL;* and *Geo infor URL*), the M&S mashup will run. The messages follow the wired boxes flow, and each box runs its function when receiving input

events and shows the corresponding visualization view in the workspace. Fig. 7 shows its execution view in Google Chrome. We can see that the MSaaS Boxes of data collection, model simulation, and results parsing have been executed successfully, and a KML with simulation results was generated and shown in the Google Map with a timeline control (in which user can choose time to predict the land use population). In addition, this mashup also shows the information of city address, weather forecast in following 5 days and Wikipedia with *North Carolina*. From this case study, we can see that anyone with basic web development knowledge (like HTML/CSS/JS) can easily develop boxes. For developing M&S mashup, no specific knowledge is required. User can simply select boxes and wire them as mashup.

7 Conclusion

We presented a new method for building 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 (MAMSaaS), a layered architecture to deploy and identify M&S mashup component as well as link and execute mashups for quick M&S application development. M&S Mashup components are called boxes, which consists of MSaaSs, Web APIs, widgets and operators. Each box has its own structure, function and view. M&S Mashup is created through box wiring mechanism. We developed tools for developing boxes, wiring boxes and run mashups. We presented a prototype with GIS M&S mashup application, which has shown that the proposed method using boxes and wiring can create and run simulation mashup in an easy and rapid way. The future work includes developing more M&S mashup applications using the proposed architecture and tools. This is an ongoing project, in order to further test its simplicity for developing M&S applications, we are going to let graduate students in our M&S course to develop their own boxes and mashups using the proposed tools. Another work is to study more on the operator boxes to handle semantic issues between boxes.

References

1. Byrne, J., Heavey, C., Byrne, P. J: A review of Web-based simulation and supporting tools. Simulation modelling practice and theory, 18(3), 253-276 (2010)
2. 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)
3. Al-Zoubi, K., Wainer, G: RISE: A General Simulation Interoperability Middleware Container Journal of Parallel and Distributed Computing, 73(5), 580–594 (2013)
4. Balasubramaniam, S., Lewis, G. A., Simanta, S., Smith, D. B: Situated software: concepts, motivation, technology, and the future. IEEE Software, 25(6), 50-55 (2008)
5. Gebhardt, H., Gaedke, M., Daniel, F., Soi, S., Casati, F., Iglesias, C.A., Wilson, S: From mashups to telco mashups: a survey. IEEE Internet Computing, (3), 70-76 (2012)
6. Papelis, Y., Madhavan, P. Human Behavior. M&S Fundamentals, 271 (2010)
7. Zeigler, B. P., Praehofer, H., Kim, T. G: Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems. Academic press (2000)

8. Skoogh, A., Perera, T., Johansson, B: Input data management in simulation–Industrial practices and future trends. Simulation Modelling Practice & Theory, 29, 181-192 (2012)
9. Fortmann-Roe, S: Insight Maker: A general-purpose tool for web-based modeling & simulation. Simulation Modelling Practice and Theory, 47, 28-45 (2014)
10. Tsai, W. T.; Fan, C.; Chen, Y.; Paul, R: DDSOS: A dynamic distributed service-oriented simulation framework1. In Proceedings of the 39th annual Symposium on Simulation. Washington, DC (2006)
11. Brebner, P: Service-Oriented Performance Modeling the MULE Enterprise Service Bus (ESB) Loan Broker Application, In Proceedings of 35th Euromicro conference on Software Engineering and Advanced Applications. Patras, Greece (2009)
12. Smit, M.; Stroulia, E: Simulating Service-Oriented Systems: A Survey and the Services-Aware Simulation Framework. 6(4), 443-456 (2013)
13. Lopes, C. V., Debeauvais, T., Valadares, A: Restful massively multi-user virtual environments: A feasibility study. In Games Innovation Conference (IGIC), 2012 IEEE International. Rochester, NY (2012)
14. 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. Computers and Electronics in Agriculture, 87, 14-18 (2012)
15. Cayirci, E: Modeling and Simulation as a Cloud Service: A Survey. In Proceedings of the 2013 Winter Simulation Conference. Savannah, GA (2013)
16. Garg. S.K., Versteeg, S., Buyya, R: SMICloud: A Framework for comparing and ranking Cloud services. Fourth International Conference on Utility and Cloud Computing. Melbourne, Australia (2011)
17. Taylor, S.J.E, Khan, A, Morse, K, Tolk, A, Yilmaz, L, Zander, J: Grand Challenges on the Theory of Modeling and Simulation. In Proceedings of the Symposium on Theory of Modeling and Simulation. San Diego, CA (2013)
18. Jung, W., Kim, S. I., Kim, H. S: Ontology modeling for REST Open APIs and web service mash-up method. In 2013 International Conference on Information Networking (ICOIN). Huket, Thailand (2013)
19. Siriwardena, P: Advanced API Security. SpringerApress (2014)
20. Lizcano, D., Soriano, J., Reyes, M., Hierro, J. J: A user-centric approach for developing and deploying service front-ends in the future internet of services. International Journal of Web and Grid Services, 5(2), 155-191 (2009)
21. Wang, G., Yang, S., Han, Y: Mashroom: end-user mashup programming using nested tables. In Proceedings of the 18th World Wide Web Conference. Madrid, Spain (2009)
22. Aghaee, S., Pautasso, C: End-user programming for web mashups. In Current trends in web engineering, 347-351. Springer (2012)
23. Trinh, T. D., Wetz, P., Do, B. L., Anjomshoaa, A., Kiesling, E., Tjoa, A. M: Open Linked Widgets Mashup Platform. In 2014 ESWC. Grete, Greece (2014)
24. Le Phuoc, D., Polleres, A., Morbidoni, C., Hauswirth, M., Tummarello, G: Rapid semantic web mashup development through semantic web pipes. In Proceedings of the 18th World Wide Web Conference. Madrid, Spain (2009)
25. Zahariadis, T., Papadakis, A., Alvarez, F., Gonzalez, J., Lopez, F., Facca, F., Al-Hazmi, Y: FIWARE Lab: Managing Resources and Services in a Cloud Federation Supporting Future Internet Applications. In 2014 UCC. London, UK (2014)
26. Wang, S., Wainer, G: Semantic Selection for Model Composition using SAMSaaS. In Proceedings of Symposium on Theory of Modeling & Simulation. Alexandria, VA (2015)
27. Wang, S., Wainer, G: Web-based simulation using Cell-DEVS modeling and GIS visualization. Modeling & Simulation-Based Systems Engineering Handbook, 3,425-467 (2014)