SYSC 5900 System Engineering Project

# Integration of RISE with Cell DEVS Web-Viewer

Submitted to:

Prof. Gabriel Wainer

Department of Systems and Computer Engineering

Carleton University
1125 Colonel by Drive
Ottawa, ON. K1S-5B6 Canada
gwainer@sce.carleton.ca

By:

Arnav Kaushal

Student No. 101114976

Department of Systems and Computer Engineering

Carleton University
1125 Colonel by Drive
Ottawa, ON. K1S-5B6 Canada
arnavkaushal@cmail.carleton.ca

## Table of Contents

## LIST OF FIGURES

Directions for use:

- Link of the actual tool:
  https://simulationeverywhere.github.io/CD-WebViewer-2.0/index.html
- Link of GitHub repo:
  https://github.com/SimulationEverywhere/CD-WebViewer-2.0/tree/DEV-Arnav
- Link of example how-to video
  https://youtu.be/n-W-AlrHlI4

  (For reference, link of my personal GitHub repo:
  https://github.com/arnavk2319/CD-WebViewer-2.0)

**STEPS TO BE FOLLOWED**

STEP 1: Go to the URL of *Link of the actual tool* via a browser.

STEP 2: Click on the cloud icon on the top left of the dashboard.

STEP 3: From the list of models that appear in a popup, select one model.

STEP 4: After the model results are loaded successfully, use the *media controls* or different components of the viewer to control the visualization of the results.

(REFER *Section 3.2.2* titled *Results* in this report OR the *Link of example how-to video* for further information)

# 1. INTRODUCTION

## 1.1 *Motivation*

Modelling and Simulation are extensive procedures that require a ton of resources for a successful simulation to happen with results that can be worked upon. Simulation is a one-of-a-kind process which happens in stages and the proper implementation of each phase is important for the entirety of the procedure. But this whole procedure requires resources like hardware compatibility, configuration of software and dependencies needed by the simulation, all-time access to the model files and simulation meta-data etc.

The existing version of the CD Web-Viewer relies on the hardware capability of the user's local system and their access to the simulation resources in the form of model files and software dependencies in order to successfully visualize the simulations successfully. Cell DEVS simulations utilize the generation of many log files, atomic models etc. files that help in the successful visualization of the entire Cell-DEVS model. The motivation for this research is presented in the form of integrating the RISE (RESTful Interoperability Simulation Environment) platform with the existing version of the CD Web-Viewer resulting in the increased scope of use for both the Cell-DEVS Web-Viewer and the RISE Platform in the direction of better modelling and simulation of the DEVS models.

## 1.2 *Objective*

The objective of this work is this work is to successfully integrate the RESTful Interoperability Simulation Environment (RISE) platform as a functionality with the existing version of the Cell DEVS Web-Viewer.

This integration would lead to the effective utilisation of the services offered by the RISE platform as a distinct functionality that will allow users to perform activities related to the simulations of their respective models using the Cell-DEVS Web-Viewer.

## 1.3 *Contribution*

With the successful completion of the objective, the following mentioned would be the contributions of this work:

1. Integrating the visualization of the Modelling and Simulation environment with the RISE platform using Web Services components.
2. Allowing users to run simulations of their model using the RISE platform and then visualize the results of the simulations on the CD Web-Viewer.
3. Visualizing several pre-loaded models on the RISE platform.
4. Interacting with the RISE platform without knowing the technical know-hows of the operational logic behind Web Services.
5. Leading to on-the-go simulation, without the need to worry for the hardware compatibility of the user's local system or the all-time access to the input simulation files for running the simulation successfully along with the effective use of the Cell DEVS Web-Viewer.

# 2. Background and Related Work

## 2.1 *DEVS Formalisms*

Due to the rise in the complex nature of artificial systems in the modern world, it has become increasingly difficult to test these systems on a large scale. Performing testing and simulation on these complex systems can be infeasible, massively time consuming and in some cases, dangerous too thus it makes this procedure tougher than it needs

to be DEVS [1] presents techniques and formalisms to simulate these complex systems using the concept of atomic and coupled models. It shows the whole system as comprised of these models in which each of them are characterized by their own variables, dependencies and which are organized into hierarchical manner so that the testing these systems can be done in a coordinated manner with every component functioning in its own way. This manner enables to model and simulate entire complex systems at once where one or more components may behave differently during different events and thus, the entire system will be affected even if one component malfunctions. This verifies the credibility of DEVS techniques and provides a feasible, safe and detailed testing aspect for the entire

system. In this work, DEVS has been used to model and simulate the working of a communication system by providing inputs like; the messages generated by each node, the position of each node and the state of the communication medium, that is, active or broken, that each node in the system uses.

### **2.1.1** *Classic DEVS*

DEVS [1] constitutes a structure for formal modelling and simulation of real-time systems by characterizing the various components of the system into different types of models each having its own parameters and thus their own manner of functioning. In DEVS, the most basic unit of a system is an atomic model which has its own parameters. Each atomic model comprises of different ports for receiving and sending data as these atomic models may or may not be connected to one or more atomic models. Coupled models consist of two or more atomic models and they also have ports which are connected to that of the atomic models constituting them.

From Fig. 1, each atomic model comprises of an input port(x), an output port(y) and a state(s) which informs whether the model is active or passive. The behaviour of the atomic model is characterized by transitions which decide the value of the state, determine the output and count in the time factor that advances progressively for the concerned simulation. External transition function $\delta_{ext,}$ is triggered whenever the model receives an external input and maps a new state from the current state using the value of the input received, the time elapsed until now and current state of the model. After that, internal transition $\delta_{int}$ is triggered, that maps the previous state value to the new one after a specified amount of time has been elapsed.

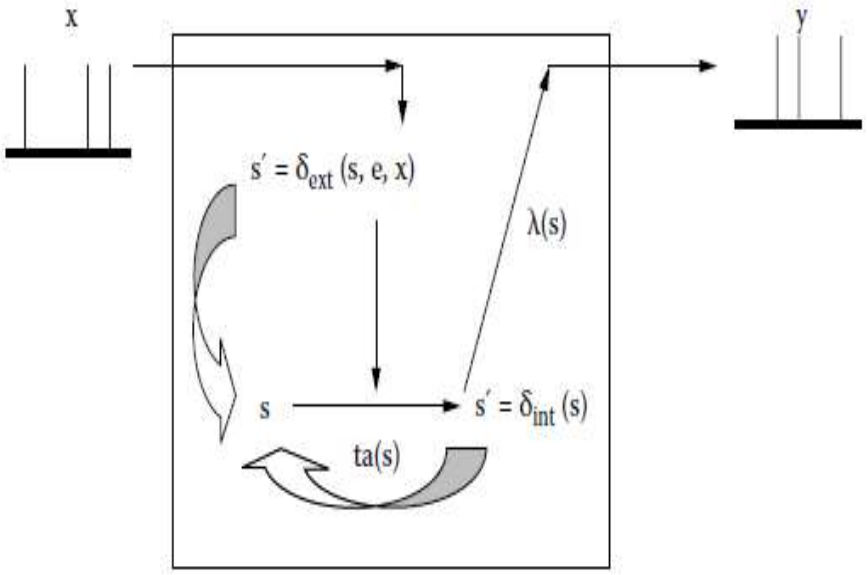


Figure 1. Informal Definition of an atomic model

The time advance function changes the current state value to the old state after a definite amount of time has passed as it determines the duration of the state held by the model. The output function gives a value to be outputted to the output port according to the state value at the time. The output value leaves through the output port of the atomic model and if this model is connected to an atomic model, this output will act as input to the other atomic model.

A DEVS coupled model comprises of two or more atomic models as shown in fig.2. These atomic models behave as described previously and can have different parameters as well. But the whole model acts as one thus, depending on the functioning of the atomic models inside. The atomic models are characterized by internal and external couplings both to each other and the external links and port, respectively. The transitions in a coupled DEVS model depend on the individual transitions taking place in the atomic model inside the coupled model.

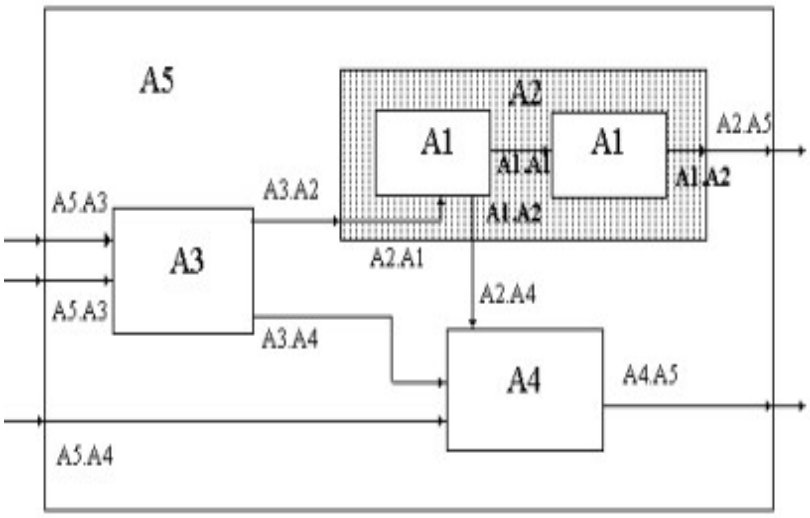


Figure 2. Coupled Model

The related work done in the field classic DEVS includes further enhancements to the existing techniques of modelling and visualization. This research shows how systems how systems can be advantageously represented as discrete event models by using DEVS (discrete event system specification), set-theoretic formalism which can be applied to advanced robotic and intelligent automation, showing how classical process control can be readily interfaced with rule-based symbolic reasoning systems [3]. This work represents a methodology which can be used to map hierarchical, modular discrete event models onto distributed similar architectures which can be used for top-down model development, expressed with the extended formalism [5].

### 2.1.2 *Cell DEVS*

Cell DEVS is a combination of DEVS and Cellular Automata which allows for implementation of cellular models with timing delays. A Cell-DEVS model can be used to describe real systems that can be represented as cell spaces. The states of the lattice are updated according to a set of rules, which involves both direct and macro implementations of those rules, thus changing the states of the cells in a synchronous and semantic manner. Each model is described as a set consisting of a time base, inputs, states, outputs, and functions [6]. A model uses input and output ports to communicate with the others. The internal and external events produce state changes, whose results are spread through the output ports. The influences of the ports determine if these values should be sent to other models.

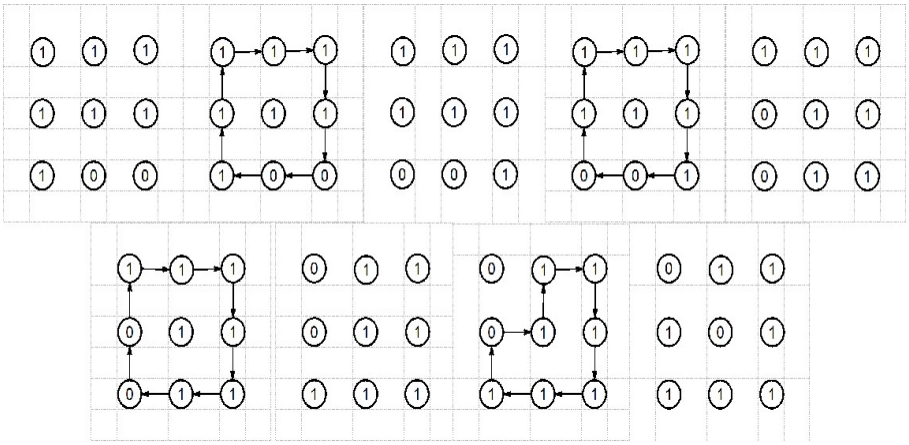

Figure 3. 2-Dimensional Cell Lattice

As shown in fig. 3, the states of each of the cells in the figure are determined by the following:

- Inputs to a given cell
- Inertial or Transport Delay
- Computation of a local function
- Outputs, due to a change in the state of a cell

Every cell includes a quantimeter value produced by the local computing function is quantized. The quantized value is compared with the quantum threshold. Two kinds of delays can be defined: transport delays model a variable commuting time, and inertial delays, which have pre-emptive semantics (scheduled events can be discarded if the computed value is different than the future state). If the threshold bound was reached, an output is provided. The output is delayed using transport or inertial delays. If the threshold was not reached, the change is not sent to other models [7]. The research done using the concept of Cellular DEVS includes interesting results. This research introduces a new high-performance simulation engine for large scale cellular DEVS models which only considers active cells during the simulation and implements a data structure which permits efficient searching of the active cells in a cell lattice [2]. The results of this experiment show the modelling and simulation of forest fire spread using Rothermel's mathematical model for fire spread [12].

### 2.1.3 *Web Development*

The implementation of the functionality in this work has been done using JavaScript and extensions of existing JavaScript libraries using the ECMAScript Version 6 or ES6 specification or JavaScript 6, commonly called, which increased the efficiency of the task at hand.

The benefits of the ES6 version over the earlier versions and mostly another wildly used JavaScript version ES5 are:

- Using the “let” and “const” keywords which allow to define a block-scoped variable and for declaring a block-scoped constant, respectively, which are valid declarations in the so-called “block”.
- Ability to use computed property names without having to declare parameters/properties separately.
- Ability to define default values for directive calls which was not possible in ES5.
- Using de-structured assignment for data structures which are a lot of help in using certain parameters in the data, mostly which are large in number and sometimes optional to use.
- Employ object initializers shorthand which leads to a decreased redundancy in the codebase.
- Introduction to Arrow functions which can be used to better deal with Promises, important for dealing with Web Services like RESTful etc.

All the improved functionalities mentioned in the list above are utilized effectively in the development of the Cell-DEVS Web-Viewer and for the integration of the RISE platform with this web application.

Moreover, for implementing the intended integration of this report, a third party library called *zip.js* has also been employed which makes use of worker threads to format the data, compress it and de-compress it in non-blocking background processes, which allows for the response from the RISE platform to be prepared in a certain format and then use that formatted response to visualize the model simulation results stored in the RISE platform on the Cell DEVS Web-Viewer. This

information informs on the environment that was employed for the purposes of this report and will be built upon on the later sections of this report.

## **2.2** *Evolution of the Cell-DEVS Web-Viewer*

### **2.2.1** *Cell-DEVS Simulation Viewer*

The Cell-DEVS Web-Viewer is a client-side web application built entirely in HTML5 and JavaScript [11]. The application was built as an alternative to the desktop-application for CD++ simulation viewer, meant to offer more control over the visualization of the simulation results and a user-friendly interface for use by even a non-technical person. The main uses of this web-based application are:

1. Loading simulation results
2. Visualizing the simulation results
3. Effective control over the various perspectives of the simulation with reference to cell state chart, state frequency etc.

The first version required the input of the following files, part of the simulation results of the model, which lead to successful rendering of the simulation results:

- model file (*.ma*): It contains the simulation configuration parameters like input, output ports, grid size
- initial values file (*.val*): It contains the cell-space state
- log file (*.log*): It contains all the transitions that were made by each cell in the cell-space for the duration of the simulation.
- color palette file (*.pal*): It contains the color pallet for visualizing the simulation results.

Features [11] of the first version of the Cell DEVS Web-Viewer employed the following parsing strategies for the simulation of the model files:

- Streaming log file in chunks of 2MB using the specification in HTML5 due to extremely large sizes of the input simulation files (in GBs), with byte-wise precision.

- Storing the event-driven changes only instead of storing the entire cell-space as that would lead to exceeding the memory capacity of standard computers along with finding a workaround to traditional compression/de-compression techniques.
- Using HTML5 Canvas which was well-suited for scenes with many elements that needed to be rendered, removed or redrawn.
- Provides statistical analytical capabilities to the user in terms of cell states using the Data-Driven Documents (D3) JavaScript library.

With the following feature set and capabilities offered by the first version of the Cell DEVS Web-Viewer, that version also had some major shortcomings[11] that did not make it a successful candidate for future use, if it had remained in the form of first version only. These shortcomings are enlisted as follows:

- Absence of a modular codebase, lack of a cohesive architecture among the different components of the simulation viewer.
- Used a loose object-oriented approach where the components were strongly coupled, and separation of concerns were lacking.
- Code base was fragile, more difficult to maintain and evolve.

### **2.2.2** *Cell-DEVS Simulation Web-Viewer (Second Version)*

The second version of the Cell-DEVS Simulation viewer was developed in order to overcome the shortcomings of the first one. The improvements made to this version were done with the objective of making this web application functionally rounded in terms of the functionalities implemented by the application so that it could be used by anyone from the perspective of both the end-user and a developer/researcher working with the Web-Viewer. The main objectives of this version are stated as follows:

1. To limit the development complexity involved in reconstructing the simulation, making it easier to maintain and evolve for future purposes.
2. Following a strongly coupled object-oriented approach as shown in fig 4 [11].

3. Including a line-chart, synchronized with the playback widget, where the state of selected cells is tracked across a simulation. This feature was implemented following feedback received from users of the first version of the platform.

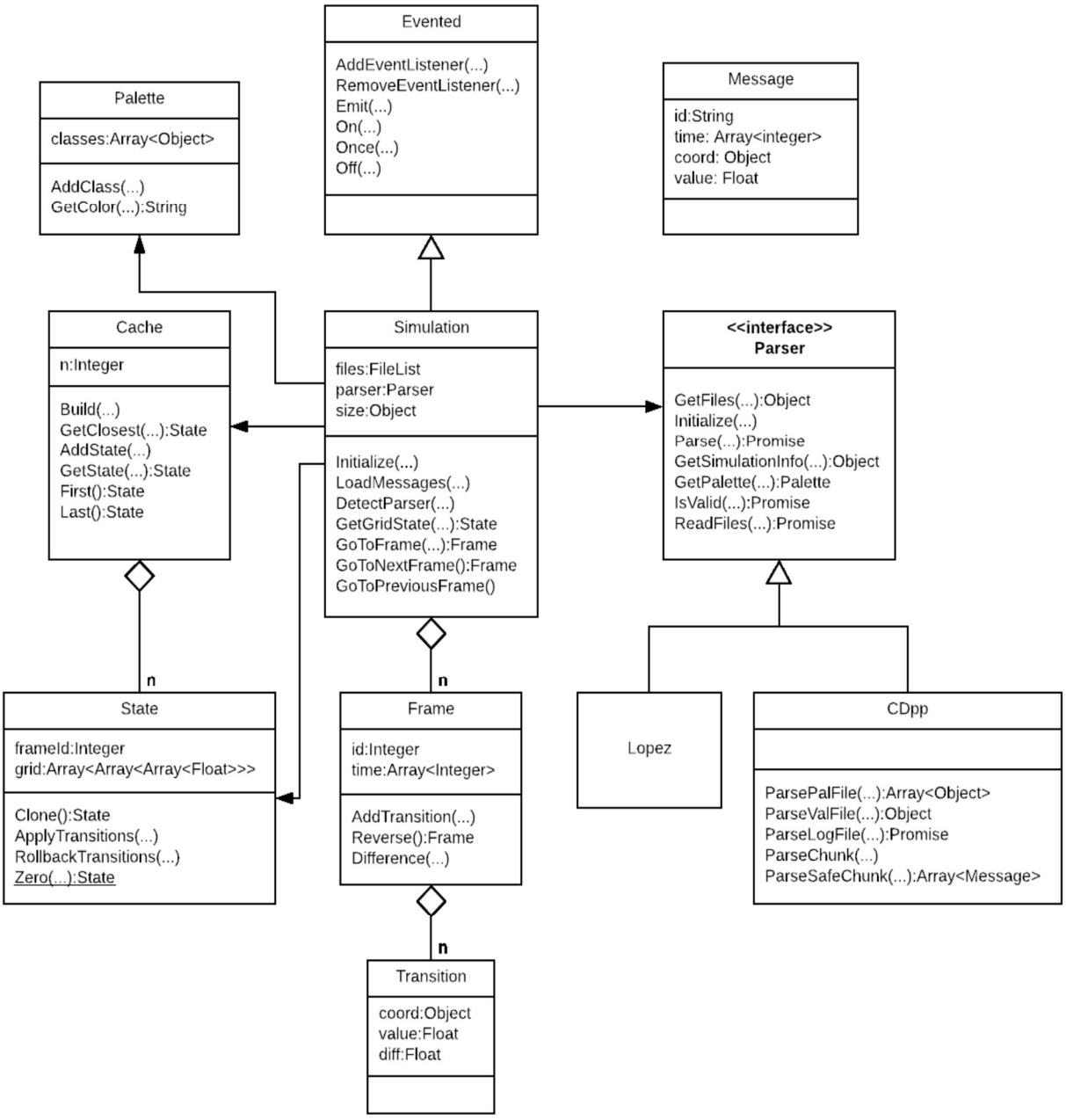


Figure 4. Strongly coupled object-oriented approach

With the strongly coupled model approach, as can be seen from fig. 4 that the main class in this approach is the *Simulation* class which holds all the data which is required to re-construct the simulation result. Even for the purposes of this report, the *Simulation* class plays a very important role for reading and parsing the *.log* file in order to render the simulation result on the Cell DEVS Web Viewer. The *Simulation* class is also responsible for going to the next frame of the simulation result as well as rolling back to the previous frame, containing the functions and classes to do the same [11].

## **2.3** *RISE Platform*

### **2.3.1** *Integrating Modelling and Simulation services with Web Services*

The RISE platform, stands for RESTful Interoperability Simulation Environment, was developed 10 years ago by Khaldoon Al-Zoubi in his work in [4], with the purpose of using it as a REST-based architecture for distributed simulation purposes [4]. The REST-based part of the architecture is what is represented by the Web Services.

Web Services, in general, represent the communication protocol that modern application uses nowadays for interacting with several types of backends, utilizing third party integrations and web hooks etc. thus making it easier to implement the different functionalities used by an application. In such a way, the application is packaged together as a bundle of functionalities that enhance the efficiency of the application as well as increase the usability and scope of the entire development process. The Web Services being employed for the purposes of this report are RESTful services which help in the successful integration of the RISE platform with the Cell DEVS Web Viewer. Some features of RESTful Web Services are:

1. **Stateless Operation:** All state management operations take place on the client side and all the client-server operations should be stateless.
2. **Layered:** The RESTful service works at the Application layer of the communication stack above all the other layers.

3. **Responsive:** The RESTful service responds back with static representations of data in various formats like JSON, XML etc. which in turn can be used by the client side.
4. **Client-Server based:** As the name suggests, this service represents the interaction between the client and server side of an application.
5. **Uniform Addressing:** The naming scheme utilized by the RESTful Web Services contain endpoints to a URI representing some data. When a request is made to that URI, the data is returned by the server in a JSON, XML format.

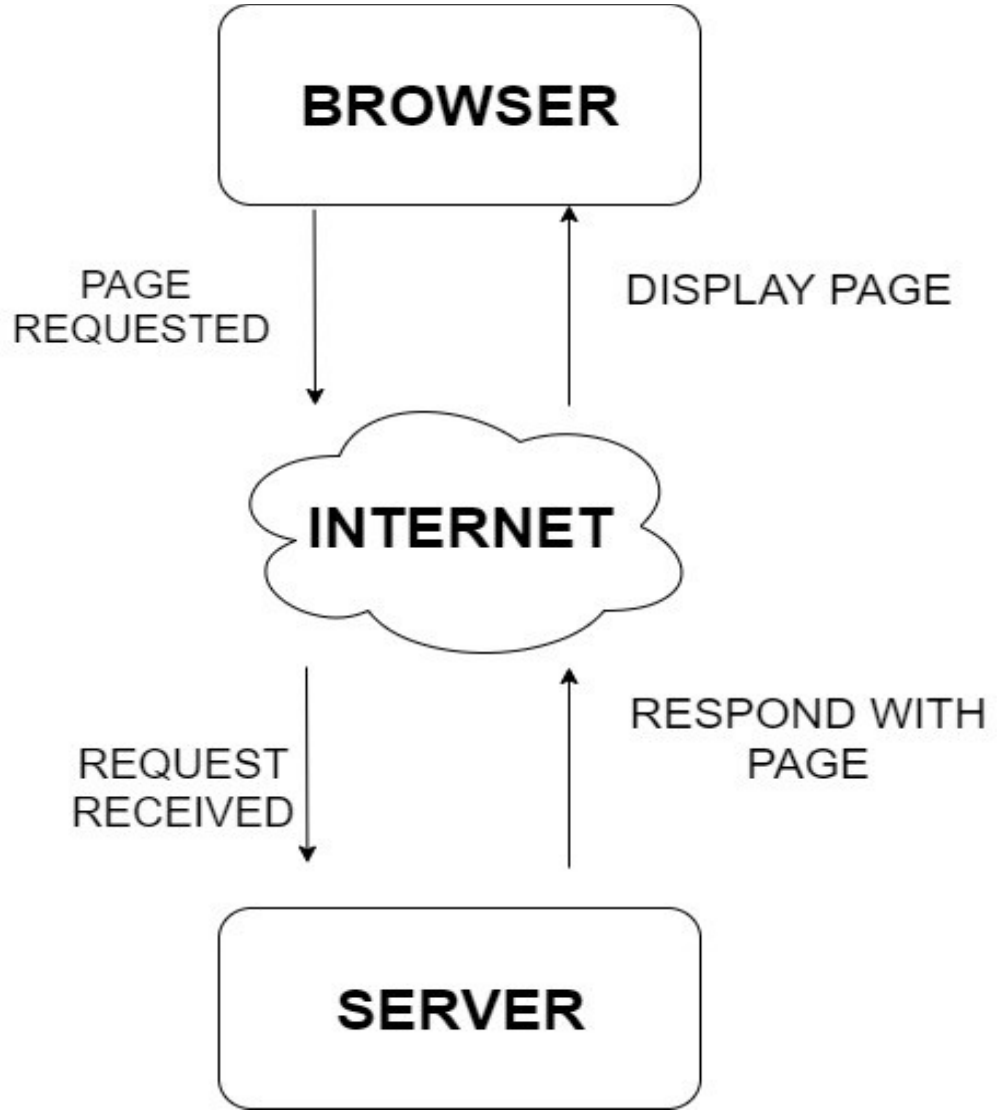


Figure 5. RESTful API

The *internet* component as seen in fig. 5 represents a list of communication protocols that are implemented at the Application, Transport and the Network layer of the modern architecture of networks. Namely, the RESTful architecture is implemented using following pre-defined methods, also acronymized as CRUD operations:

- *CREATE*: HTTP POST
- *READ*: HTTP GET
- *UPDATE*: HTTP PUT
- *DELETE*: HTTP DELETE

The client application requests an object/information from the server by making this request to a particular URI of the server, also called endpoint, thus getting the requested object in the desired format, given that the correct request parameters were included in the request. This successful interaction between the browser and the client-side application by making use of the different communication protocols is the basis behind the integration of the existing version of the RISE server with the second version of the Cell DEVS Web Viewer.

### **2.3.2** *Existing Structure of the RISE platform*

The existing structure of the RISE platform contains various CD++ versions, including **DCDpp** for normal and distributed simulation and **Extended Rules interpreter** with different port/variable values [13]. Moreover, the only way to interact with the RISE platform, before the completion of the objective of this work, was through CLI pre-made commands where in the user need to have an all-time access to the input files for the model simulation on their local machines. These pre-made CLI commands can be hard to understand with perspective to syntactical and semantic meaning to a person with a non-technical background.

RISE platform uses the RESTful services in order to interact wit the client where in CRUD operations like HTTP POST, PUT, DELETE, GET are used by the end-user who uploads the simulation files (*.ma* file, *.log* file etc.) to the RISE server where the simulation is executed and then the results of the simulation can be then accessed via the said pre-made CLI commands. Such

a kind of on-the-go modelling and simulation makes it a lot easier for any user to execute their simulation and get the results.

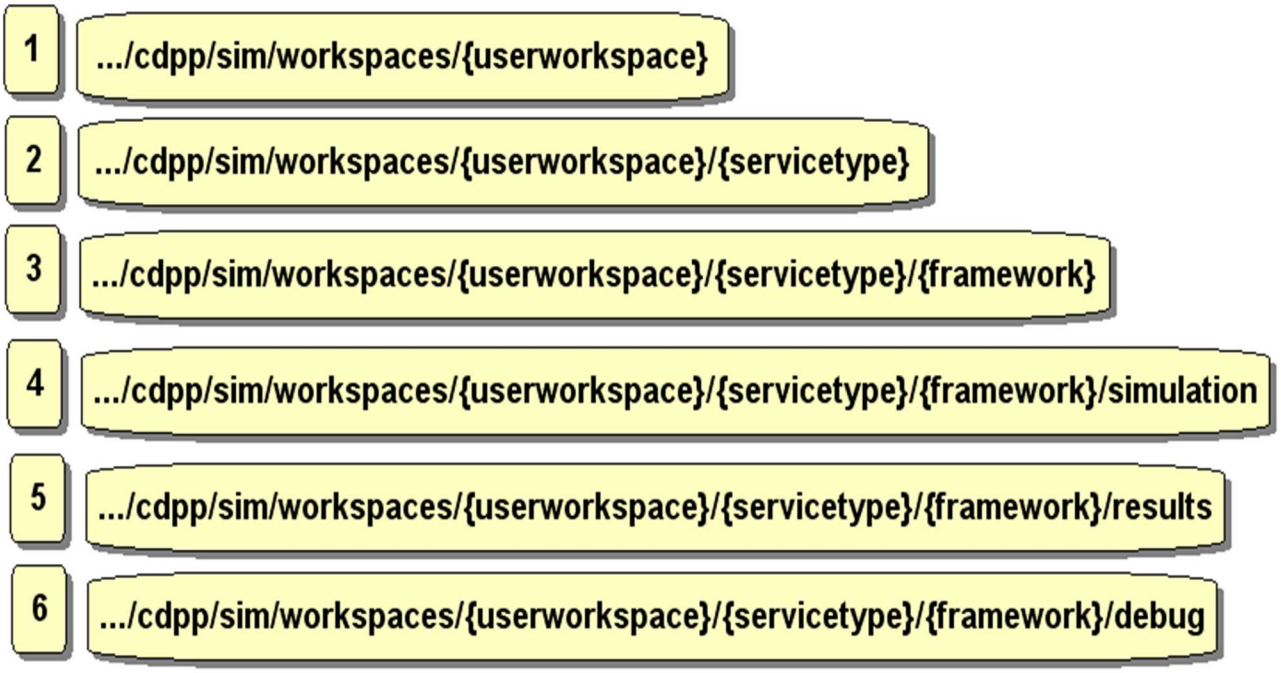


Figure 6. CLI pre-made commands

The pre-made CLI commands in the format of URLs that need to be specified by the user in the terminal as shown in fig. 6 show the syntax of the different requests, in order of the index number beside them. The first part of the URL remains same for all the requests and the phrases in curly brackets ("{}") are explained as follows:

1. *Userworkspace*: This is the name of the workspace in which the end-user is executing their simulation.
2. *Servicetype:* This is the name of the service being utilised by the end-user. These services are, namely *dcdpp* and *lopez*, belonging respectively to the normal/distributed simulation and the **Extended Rules interpreter**.
3. *Framework:* This is the name of the model that user provides in the beginning of the execution of the simulation, when providing the simulation input files.
4. *Simulation:* This command initiates the simulation activity of the user's model input files and after the user has provided the name of the model in the command 3 in the fig. 5.

5. *Results:* This downloads the simulation end results onto the local system of the user.
6. *Debug:* This allows the user to view the log file and correct any errors that occurred.

### **2.3.3** *Advantages of integrating the RISE platform and Cell DEVS Web-Viewer*

The integrations of the RISE platform with the viewer will prove to be beneficial for both. Since the Web-Viewer is used for visualization of the simulation results, integrating the RISE platform will increase the efficiency of the Viewer along with leading to an increase in the efficacy of the RISE platform as well. Some of the advantages that this integration will offer for both are:

1. The end user can do execution of their simulation input files on the go without having to worry about the hardware capability of their local system or having all-time access to the input model files.
2. With this integration, the interaction with the RISE platform becomes user-friendly and the user does not need care about the syntax/semantics of the CLI commands.
3. For a new user of the Cell DEVS Web-Viewer, they can visualize simulations of pre-loaded models from the RISE platform, verifying how the viewer works.
4. Integrating these together increases the scope of both as the functionality of the viewer is enhanced multifold due to addition of a cloud functionality and the RISE platform can now be interacted with in a much more hassle-free manner.
5. The resultant viewer, after the integration, can be developed into a modelling and simulation environment making it a one-stop environment for all model simulation activities.

## 3. Integration and Advancement

### **3.1** *Initial integration with RISE*

### **3.1.1** *RISE and CORS*

The RISE platform is integrated with the Cell DEVS Web Viewer with the help of RESTful API using HTTP operations at the application layer of the communication stack. CORS (Cross-Origin Resource Sharing) is a mechanism that is invoked when an application tries to communicate with a server using RESTful API, which operates in the manner that whenever an application tries to request resources from the server and if both of them are at different origins then CORS restricts cross-origin HTTP requests initiated from scripts. Origin here refers to the scheme(protocol), host(domain) and port of the URL used to access it.

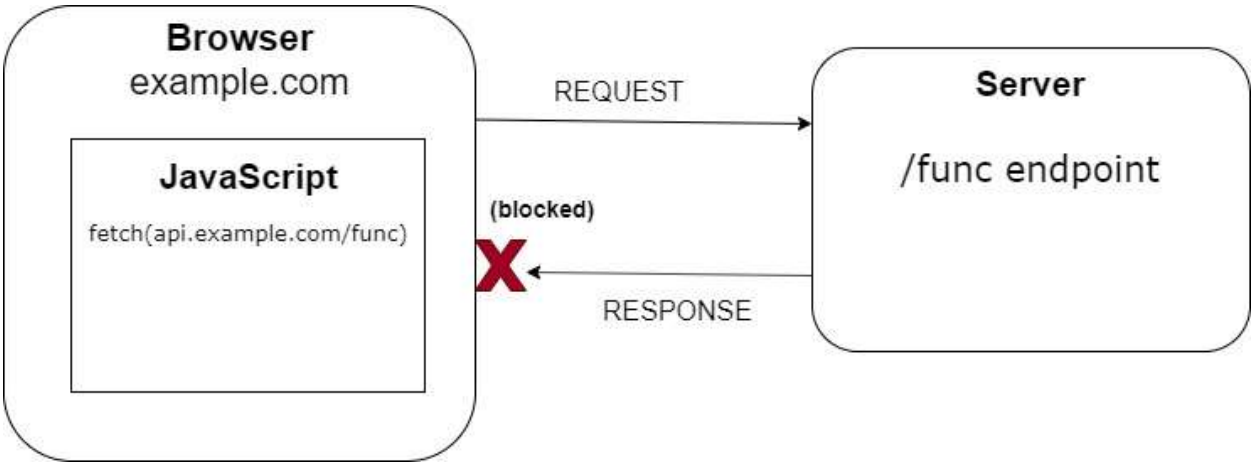


Figure 7. Cross-Origin request blocked

As shown in the fig. 7 above, the application makes a request to a particular resource using HTTP operations from the script in the application, the same scripts responsible for running the application, the server responds with an error message as shown in fig.7 thus blocking the request from the application.

Access to XMLHttpRequest at 'http://vs1.sce.carleton.ca:8080/cdpp/sim/workspaces/test/dcdpp/${modelName}/results.zip' from origin 'http://localhost' has been blocked by CORS policy: Response to preflight request doesn't pass access control check: No 'Access-Control-Allow-Origin' header is present on the requested resource.

Figure 8. CORS Error

The error shown in the fig.8 above is the example of a restriction that is provided by the browser when the application tries to access the RISE platform. For the purposes of the completion of this report, the workaround that was implemented was to use a browser with *disabled web security* parameter to true since this error is brought upon by the network security policy of the browser. Thus, disabling the network security of the browser lead to successful testing of the integration of the RISE platform and the Cell DEVS Web Viewer.

To achieve the same, the user needs to enter the following command in their terminal after making sure that all the instances of the browser are not running/killed:

```
chrome.exe --disable-web-security --user-data-dir=C:\temp
```

Figure 9. Disabled security browser

Moreover, before using this as the final measure as a workaround to the cross-origin request error were the following things:

1. Setting a header for the HTTP GET request when interacting with the RISE platform in the following manner:

   ```
   xhr. setRequestHeader ("Access-Control-Allow-Origin", "http://vs1.sce.carleton.ca:8080");
   ```

2. Trying to connect to the RISE platform from the Cell DEVS application using different origins, meaning from using different connections in order to connect with the RISE platform.

Both these measures were not directly helpful in leading us to connect to the RISE platform and therefore the final measure of using a browser with a disabled security parameter was employed as the error originates from the functioning of the browser in the first place.

### 3.1.2 *Third-party Integrations*

Since the RISE platform was created 10 years ago, JSON response and RESTful architecture were not very famous at that time therefore the response from the server was in the form of XML which could not be used directly in the Cell DEVS viewer. Moreover, the one component that can be utilized to view the simulation in the viewer is the log file (*.log* file) that needs to be obtained from the RISE platform via RESTful API. Now, the response from the RISE platform looks something like this, as shown in fig 10.

```
<html>
    <head>
        <title>RESTful-CD++ Web Services</title>
    </head>
    <body>
        <h3>LifeModel Framework State</h3>
        <h4>Description:</h4>
        <p> This model Simulates Life using Cell-Devs. It uses only one machine to do the simulation. </p>
        <h4>Restricted: false</h4>
        <a href="/cdpp/sim/workspaces/test/dcdpp/LifeModel/results/">Download Last Simulation Results</a>
        <br>
            <a href="/cdpp/sim/workspaces/test/dcdpp/LifeModel/debug/">Download Model Debugging/Statistics Logs</a>
            <br>
                <p>Time of Framework's Owner Last Activity: 2019-12-04 11:47:05</p>
                <h4>Previous Simulation Run Total Execution Time: 0.112 Seconds.</h4>
                <h4>Simulation Status: DONE (Please see descriptions below)</h4>
                <p>Simulation Status Descriptions:</p>
                <ol>
                    <li>IDLE: Simulation was never run on this framework.</li>
                    <li>INIT: Simulation is being initialized this framework.</li>
                    <li>RUNNING: Simulation is currently running in this framework.</li>
                    <li>STOPPING: Simulation is being stopped in this framework.</li>
                    <li>DONE: Previous simulation run has completed successfully.</li>
                    <li>ABORTED: Previous simulation run was aborted by Owner before completion.</li>
                    <li>ERROR: Previous simulation was exited on ERROR.</li>
                </ol>
                <h4>CD++ Options: </h4>
                <ul>
                    <li>Simulation Stop Time: 00:00:02:000</li>
                    <li>Parsing (used for debuging): false</li>
                </ul>
                <h4>Attached Model Files: </h4>
                <ol>
                    <li>life.ma</li>
                </ol>
                <h3>DCD++ Grid: </h3>
                <p>Note: Server names are automatically constructed from the machine IP address and the TCP port of the RESTful-CD++ engine.</p>
                <h4>Server Name: 134_117_62_118_8080</h4>
                <ul></ul>
                <ul>
                    <li>Cell-DEVS Zone: life(0,0)..(19,19)</li>
                </ul>
            </body>
        </html>
```

Figure 10. Response from RISE platform

The response as shown in the above figure does not inform anything about the model results as such, that is in JSON format, which would have made this directly usable instead the results are provided in the form of a URL that when looked upon, downloads the simulation results of the user's model simulation result files in the form of a ZIP file onto the user's local system. This made the integration even more difficult since the results were not directly in a usable form. When the URL is inspected using a third-party software like Postman, the response of the HTTP GET request is in the "application/zip" encoding which also made the results of the simulation from the RISE platform unusable in the present format. The "application/zip" response is shown in the figure below, fig. 11.

Figure 11. ZIP File Response

Since this response was not usable in the direct format, for the purpose of integration of the RISE platform with the Cell DEVS Web Viewer, we had to make use of a third-party library for JavaScript called *zip.js* which handles the zipping and unzipping of files and convert them into the required format. This library makes use of inbuilt functions and classes in order to convert the encoding of the files to the desired format. Some of the features of the *zip.js* library that made us use it are enlisted as follows:

1. *I/O reading capabilities*: Providing constructors to read different file types like Blob, Http, Text etc.

2. *ZIP Reading*: Allows to read ZIP file data/raw ZIP data and then convert it to the desired format, as we are doing in this integration of the RISE platform and the Cell DEVS Web Viewer.
3. *Web Worker Threads*: These threads allow for compression/decompression operations in the background in the form of non-blocking processes so that do not interfere with the main thread, on which the Web Viewer is running.

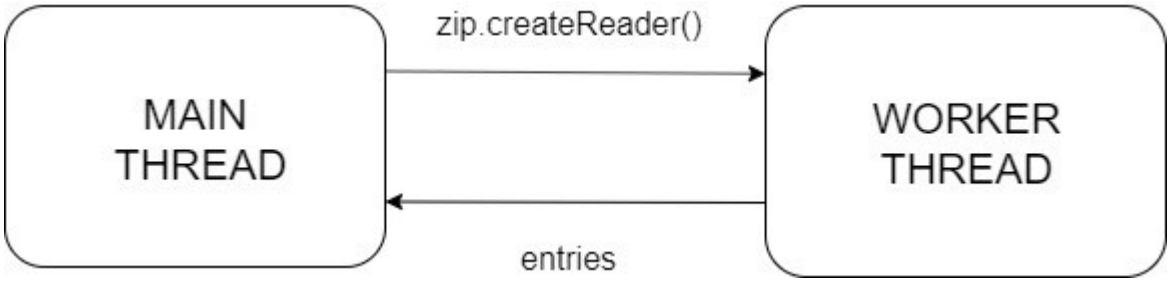


Figure 12. Web Worker Threads

As shown in fig. 12, overall there are multiple threads that are running behind the Cell DEVS web viewer which can be categorised into two categories, namely : main and worker threads where the worker threads work in the background in the form of parallel multiple threads and interact with the main thread in a non-interfering manner. The main thread is responsible for rendering the components of the Cell DEVS Web Viewer and the different functionalities implemented by it, therefore when the viewer places a request to the RISE platform, these worker threads are called into action implementing the *createReader* method that takes the ZIP file response from the RISE platform with the encoding “application/zip”.

Since the main thread is using the DOM to render the web viewer and the different components of the simulation that the user can control to change the perspective of the visualization of the results, therefore all the unzipping processes function on the worker threads after the termination of which, the results are sent back to the main thread for visualization by the end user.

### 3.1.3 *Flow Level Transition*

This section concentrates on the flow of the control in the entire Cell DEVS Web Viewer application and the steps that were performed in order to make sure that the correct parser was detected for the model simulation results so that the model simulation results could be correctly loaded into the viewer and the end-user can successfully visualize them as can be seen from fig. 13 in the next page.

Each of these steps are critical in the successful loading of the model simulation results along with the proper formatting of the simulation results obtained from the RISE platform before feeding it to the *Simulation* class of the application, which is explained in the next section. Below is given the in-detail explanation of each of the steps as illustrated in fig.13.

1. *Interact with RISE* is the beginning step where in we use the RESTful API to contact the RISE platform.
2. *Get Simulation Results* is the proper formatting of the simulation results obtained from the RISE platform.
3. *Simulation class* is the main information class in the entire application where all the data and the operations regarding the model simulation results are stored, responsible for loading/reading the results and the visualization of the results. Also, this class is responsible for the different visualization components of the viewer.
4. *Read File Contents* is the step where in the file contents of the simulation results are resolved and then converted to the require format. Th execution of this step depends on the fact whether the file contents are in the required pre-conversion format because of which there can be failure and success scenarios. Failure scenario leads to an Error *Message* while the success scenario leads to the execution of the next step.
5. *Detect Parser* is the critical step where in the type of simulation result is detected, in which either *CDpp* or *RISE* parsers are detected. RISE parser files are the simulation result files, executed on the platform itself, which are loaded directly from the RISE platform into the viewer. CDpp parser files are the simulation result files that the end user directly loads into the viewer from their local system by dragging and dropping/selecting from their local file directory.

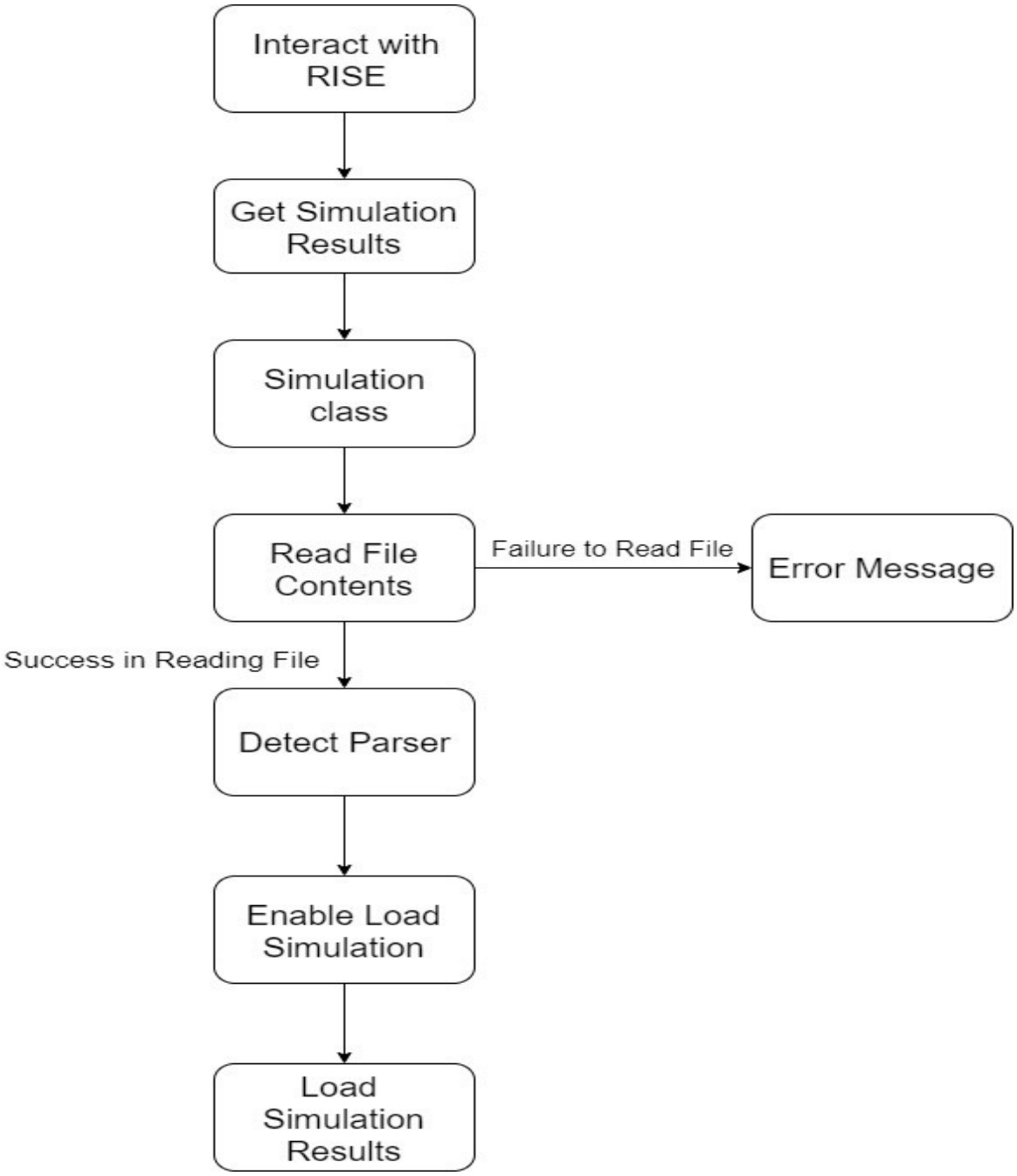


Figure 13. Control Flow

6. *Enable Load Simulation* enables the DOM element “Load Simulation” which allows the user to load the files into preview mode into the viewer.
7. *Load Simulation Results* loads the results so that the user can visualize their results using different components of the Web Viewer.

## 3.2 *Final Integration*

### 3.2.1 *Integration Approach*

Since the response of the model simulation result files from the RISE platform are in the form of ZIP file with the encoding "application/zip'. To tackle this issue and successfully resolve the response in a format that could be understood by the *Simulation* class, the following steps were undertaken, as can be seen in fig. 14:

1. *HTTP GET REQUEST*: Placing an HTTP GET request to the RISE platform using the architecture as provided by the RESTful API.
2. *Response:* The response returned by the GET request returns a ZIP folder which can be seen in the fig.10.
3. *Convert Response:* The response is converted to a BLOB (Binary Large OBject) using the *new* keyword and the inbuilt *Blob* class as the "application/zip" encoding cannot be directly used by the *zip.js* library.
4. *Unzip BLOB:* The BLOB object created in the previous step can now be used by the *createReader* method where in the *BlobReader* method in invoked by one of the worker threads running in the background in a non-blocking process. The *createReader* method allocates the unzipping process to a separate thread where in the *BlobReader* method is used by that worker thread to do the same.
5. *Parallel Worker Threads:* Each of the web worker threads work in parallel in unzipping the entire response folder containing multiple simulation result files. These multiple files can be *.log* and *.out* files in the parent folder, out of which we only need one *.log* file in order to visualize the results in the viewer.
6. *Child: LOG file:* The LOG file response is read using the *zip.js* library in which the *TextWriter* method is invoked in order to convert the file contents to plain text.
7. *Object Creation:* The desired component of the response from the RISE platform, converted to plain text now, is then used to create a tuple-type *File* object containing the name, content and the raw parameter in the array-type declaration.

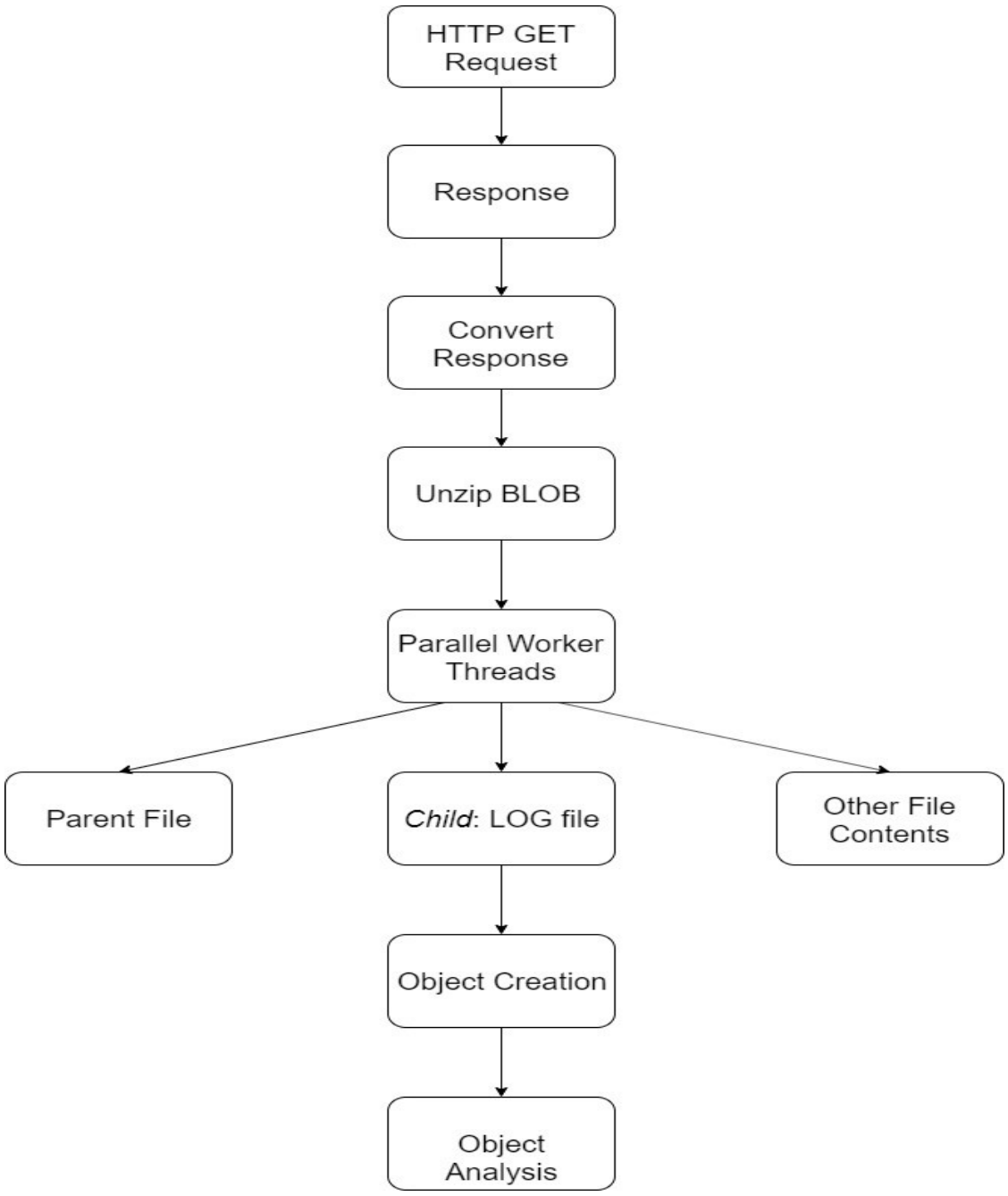


Figure 14. Response Conversion

8. *Object Analysis:* After the creation of the FILE object, the same control flow is followed as mentioned in the previous section and illustrated in fig.13. The data is then sent to the simulation class where in the parser type is detected enabling the successful loading of the simulation results in the viewer.

### 3.2.2 *Results*

This section illustrates the results of the integration of the RISE platform with the Cell DEVS Web Viewer. Each of these figures show the steps that the user needs to perform in order to interact with the RISE platform.

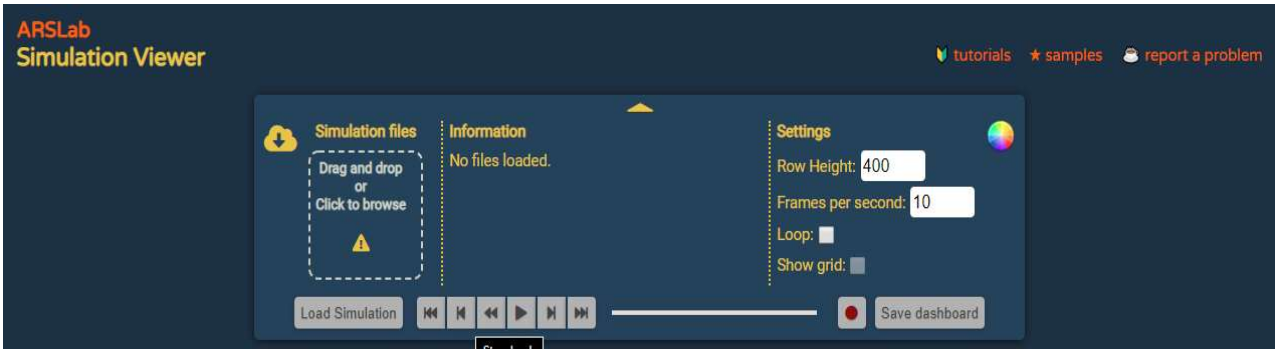


Figure 15. Index Page

As can be seen from fig. 15, the integration of the RISE platform is shown on the top-left corner of the dashboard of the Cell DEVS Web Viewer in the form of a cloud icon.

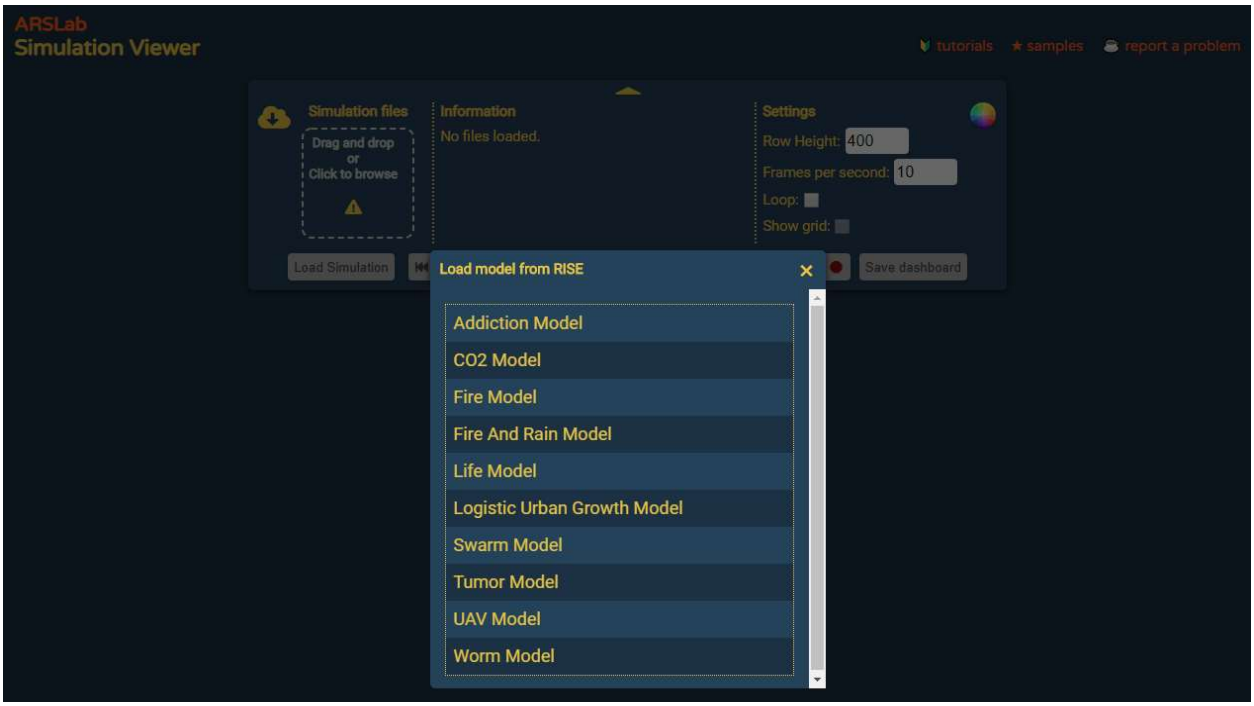


Figure 16. Pre-loaded models

After clicking on the icon, the user is presented with a popup showing the user a list of certain models as can be seen in fig.16. These models are simulated on the RISE platform and their simulation results are present on the server so that these results can be visualized on the Cell DEVS Web Viewer.


Figure 17. Simulation Loaded

Once the models are simulated on the RISE platform, their results are available on the Cell DEVS Web Viewer via the integration done in this paper. Clicking on one of those models enables 'Load Simulation" button which is disabled by default. Once enabled, clicking on that button loads the simulation results successfully as can be seen in fig. 17. The *Simulator* parameter shows the type of model simulation that was done, functioning in the same way if the user had chosen the files from their local system.

Moreover, the *Files* parameter, in the dashboard, show the file of the simulation results as loaded from the RISE platform. These names are auto generated by the platform and the user cannot make any changes here.

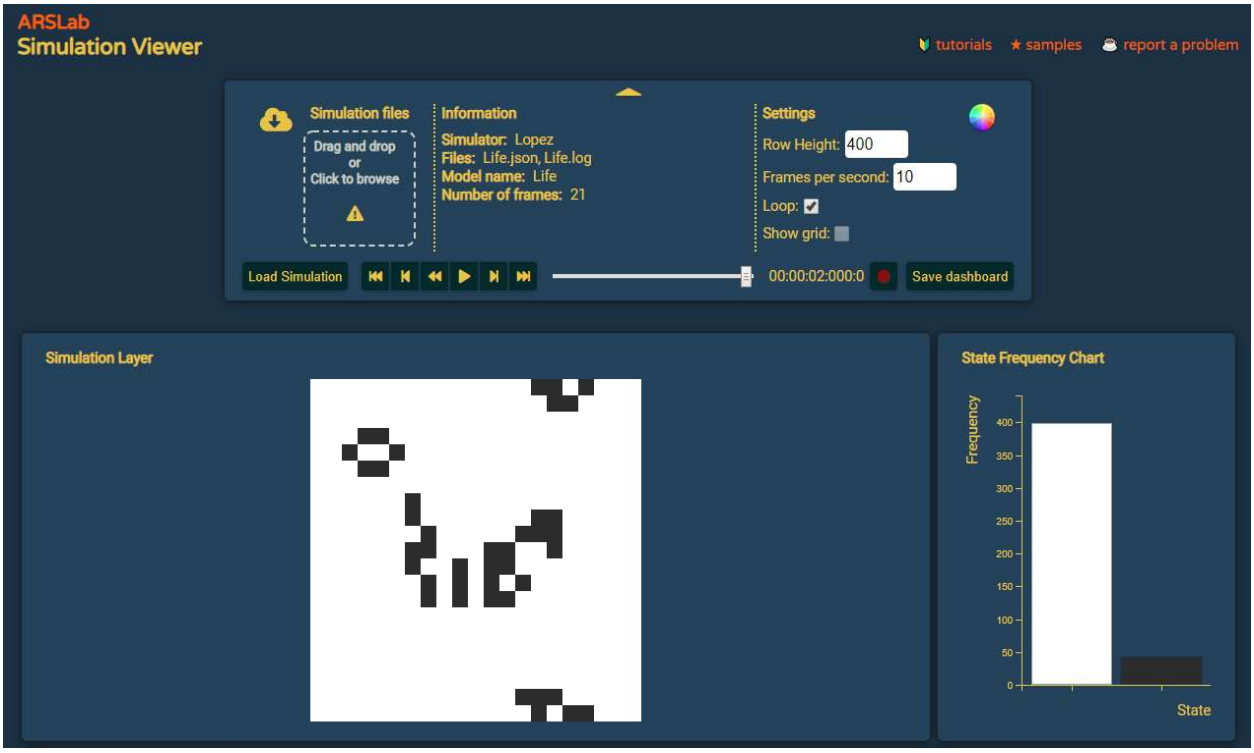


Figure 18. Visualizing simulation results

After the models are loaded successfully, the user can make use of certain components of the viewer, like the *media control* buttons or the *pallet* etc. to control the way they want to visualize their results, as shown in fig. 18. Other components present in the viewer like the *Cell Track Chart*, *State Frequency Chart, Transition Heat Map* depending on the type of models that were simulated.

# 4. Conclusion

## 4.1 *Summary*

The Cell DEVS Web Viewer provides the functionality of visualizing the model simulation results in a detailed manner, consisting of components that the end user can utilize, to do the same. In addition to that, the RISE platform provided on-the-go simulation without having to worry about the hardware capability of their local systems but the all-time access to the model simulation files and the intricate way in which the RISE platform was interacted with, explained in the previous sections, made it arduous ,for the different kind of users, to interact with. This work aimed to

bridge that gap, by integrating the RISE platform with the Cell DEVS Web Viewer to provide a one-stop simulation and visualization application where in the end-user, being from different fields, can easily make use of this application along with the RISE platform to successfully simulate and visualize their model simulation results. To do this integration, different tools and third-party integrations were employed in addition to the different approaches that were adopted, which therefore lead to the completion of the mentioned objective.

## 4.2 *Future Work*

There is certain scope of future work that can be done in order to enhance the functionalities of the web application along with that of the RISE platform. Some of these are:

1. Building visualization tools that rely on the RISE platform in the form of embeddable viewers, visualization APIs etc.
2. Updating RISE for modern browsers since it currently operates only on one, like adding specific headers on exchanged requests, authentication, parameter queries involving HTTP GET and POST, and responses for non-standard requests involving HTTP PUT and DELETE.
3. Finding permanent solutions to Cross-Origin Resource Sharing (CORS) thus making the RISE platform easy to access via different platforms.
4. Updating the technical stack behind RISE as it currently runs on Restlet, which was the first ever REST framework built for Java, to more modern frameworks.
5. Adding more functionalities like changing the response of the server from XML to JSON, even for the model simulation results, will allow RISE to be integrated much more deeply in the Cell DEVS Web Viewer, adding repository exploration functions and parsing results.
6. Creating a reusable DEVS API that would include data structures to hold the data, additional User Interface components, communication with the RISE platform and visualization of results.

# REFERENCES


[1] B. Zeigler; T. Kim; H. Praehofer: Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems, Academic Press, 2000.

[2] Hu, X. and Zeigler, B.P., 2004, April. A high-performance simulation engine for large-scale cellular DEVS models. In *High Performance Computing Symposium (HPC'04), Advanced Simulation Technologies Conference* (pp. 3-8).

[3] Zeigler, B.P., 1989. DEVS representation of dynamical systems: Event-based intelligent control. *Proceedings of the IEEE*, *77*(1), pp.72-80.

[4] Al-Zoubi, K. and Wainer, G., 2009, June. Using REST web-services architecture for distributed simulation. In *Proceedings of the 2009 ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation* (pp. 114-121). IEEE Computer Society.

[5] Concepcion, A.I. and Zeigler, B.P., 1988. DEVS formalism: A framework for hierarchical model development. *IEEE Transactions on Software Engineering*, *14*(2), pp.228-241.

[6] Wainer, G., 2000. Improved cellular models with parallel Cell-DEVS. *Transactions of the SCS*, *17*(2).

[7] Alejandro López, Gabriel Wainer: Extending CD++ Specification Language for CellDEVS Model Definition (DRAFT. Not for distribution)

[8] Al-Zoubi, K. and Wainer, G., 2011. Distributed simulation using restful interoperability simulation environment (rise) middleware. In *Intelligence-Based Systems Engineering* (pp. 129-157). Springer, Berlin, Heidelberg.

[9] Al-Zoubi, K. and Wainer, G., 2013. RISE: A general simulation interoperability middleware container. *Journal of Parallel and Distributed Computing*, *73*(5), pp.580-594.

[10] Al-Zoubi, K. and Wainer, G., 2010, December. Rise: Rest-ing heterogeneous simulations interoperability. In *proceedings of the winter simulation conference* (pp. 2968-2980). Winter Simulation Conference.

[11] St-Aubin, B., Hesham, O. and Wainer, G., 2018, July. A Cell-DEVS visualization and analysis platform. In *Proceedings of the 50th Computer Simulation Conference* (p. 14). Society for Computer Simulation International.

[12] Ntaimo, L. and Khargharia, B., 2006. Two-dimensional fire spread decomposition in cellular DEVS models. *Simulation Series*, *38*(1), p.103.

[13] RISE User's Guild Manual (Integrated version), Sixuan Wang, Gabriel A. Wainer