BUILDING ADVANCED VISUALIZATION

Shreya Taneja

Department of Systems and Computer Engineering Carleton University

1125, Colonel by Drive

Ottawa, ON. K1S-5B6 Canada shreyataneja@cmail.carleton.ca

***Abstract*—**This paper describes the advance visualization of Cell DEVS Viewer. The idea is to improve the WebViewer by integrating sophisticated and user-friendly visualization options into the JavaScript Cell DEVS Viewer. The WebViewer is engineered to visualize the results of Cell DEVS and Classic DEVS simulations. The application is designed using JavaScript and HTML5. The major focus will be defining advanced visualization of DEVS and Cell DEVS models. The major purpose of the WebViewer is to increase user interactivity and improved feedback for users. The paper focuses on updating WebViewer to display more precise and accurate results.

Keywords*—DEVS, Cell-DEVS, Visualization, WebViewer.*

Introduction

The Cell-DEVS WebViewer is a minimalistic client-side web application built entirely in HTML5 and JavaScript. The web applications are always a subject of improvements. Many kinds of improvements are needed in a web application time-to- time to meet user requirements, providing new features to the users, etc. That is why a new feature has been introduced to the WebViewer in the current version, where a user can visualize the results of Classic DEVS Models using various Visualization tools in form of charts, bar graphs or diagrams. As this is the newly added feature in the WebViewer, it is subjected to many improvements based on the reviews from users or other developers. In this paper, we discuss about some of the required improvements that have been made to the latest version of WebViewer to improve user interaction for the DEVS models.

The Cell DEVS WebViewer is designed to visualize the results of classic DEVS and CELL DEVS models. There are many applications available to design the models but none of them focuses on the visualization of the models. This application helps the user

to visualize their model for better analysis of the design. The user can view and monitor the result using a single stand-alone application where he can view their simulations as well as analyze the outputs of different states using different analytical bar graphs, heat maps and line charts. This gives an advantage to user to analyze different aspects of the model before conducting a real-life experiment. Technologies being used during the project is mainly HTML5, CSS, JavaScript and D3. WebViewer application follows an Object-Oriented approach, which makes it easier to add new features or update the already existing features. This project emphasizes more on refinement of the User Interface and on some parts of the visualization of classic DEVS models.

Classic DEVS are represented visually (DEVS diagram) using Scalable-Vector Graphics (SVG), which is easy to animate and modify using JavaScript. This SVG should animated and various features are added to the code to enhance the user experience and accessibility.

The tool is also integrated with RISE sever. RISE is a backend simulator and library. RISE gives user accessibility to remotely manage the complete lifecycle of a simulation. Through which user is able to create a workspace in the remote server for the simulation. From where he can download the results for the simulation after uploading certain file. This saves user time and money as he does not need to buy any expensive computation resources to perform the simulation and also it removes dependency of user for installing applications to run the simulation.

Background

DEVS Overview

With the advancements in the modern world there is need to design complex architecture to meet the requirements of modern world engineering. To design or perform these complex experiments, it requires a lot of resources, the nature of these experiments and artificial systems can be highly complex. These experiments a simulated virtually before implementing them in real life. It saves time and money of the user and gives user advantage to reconfigure or redesign the model after analyzing the outputs.

There are different states in case of modeling and simulation. The simulation process begins with the practical problem that requires solving and understanding. Then a conceptual model is defined which gives a high level of the structures, models behavior and input output is given. Then the input/output data is collected and is analyzed based on attributes while designing the conceptual model. Based on the data and the conceptual model the model is built with all the attributes and methods. Then the simulation algorithm is defined and is translated into a computer program. Then the verification and validation of the model is done. Then the outputs of the model are experimented using a simulator and are verified with the goals stated in the conceptual model. The system behavior is then understood based on the simulation outputs.

Discrete Event Modeling and Simulation (DEVS) is based on general system theory and it allows to represent all the system whose input/output behavior can be

described in sequence of events. DEVS follow a hierarchical and modular nature. DEVS constitutes a structure for modeling and simulation of real-time systems categorizing the different components of the system into different models each having their own input/output and behavior.

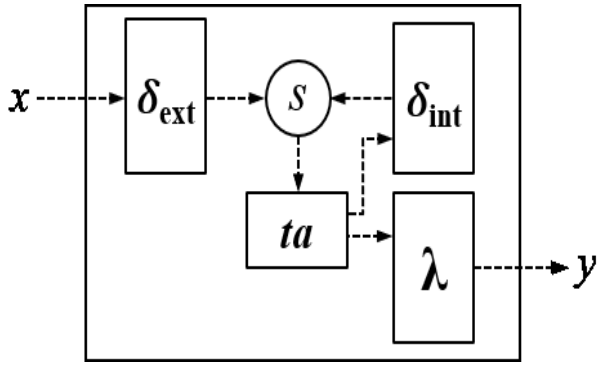

Figure 1

DEVS can be described as a composition of atomic and coupled models. Basic models (called atomic) are specified as black boxes which comprises of inputs and outputs. Each atomic model has a persistent state. As soon as the model receives an input an external transition function is triggered, and the state of the model is changed. The output function will also generate an output value based on the current state before triggering the internal transition function. The time for internal transition is specified by time advance which is also dependent on the current state. The internal transition function will update the state and new time advanced is calculated. The model will remain passive until new external input. A coupled model is formed by carefully integrating the input and output of two atomic models.

Cell DEVS Overview

Cellular automata is a well know way that describes the system that can be represented as cell spaces. In cellular automata cells can take finite values. States in the lattice are updated according to a local rule in a simultaneous and synchromesh way.

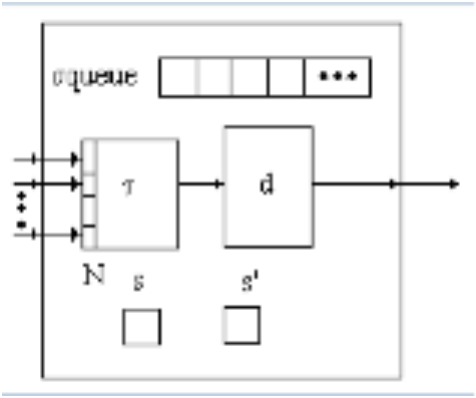

Figure 2

In Cell-DEVS each cell in the lattice is defined as an DEVS Atomic model and the whole cell space can be built as a DEVS coupled model.

A cell uses a set of N input values to compute its future state, which is obtained by applying the local function t. A delay function is associated with each cell, after which, focus on the behavior of the t and delay functions After the basic behavior for a cell is defined, a complete cell space can be built as a coupled Cell-DEVS: the new state value is sent out. A coupled Cell-DEVS is composed of an array of atomic cells, each of which is connected to the cells in the neighborhood. The border cells (B) can be provided with a different behavior than the rest of the space. The Z function defines the internal and external coupling of cells in the model. The X list and Y list are used for defining the coupling with external models. The CD++ tool implements both DEVS and Cell DEVS theories. The tool is built as a hierarchy of models, each of them related with a simulation entity. Atomic models can be

described using a graphical notation. A specification language allows defining the model's coupling, including the initial values and external events. The tool includes an interpreter for a specification language that allows describing Cell-DEVS models. The behavior specification of a Cell-DEVS atomic model is defined using a set of rules with the form: POSTCONDITION ASSIGNMENTS DELAY PRECONDITION. These indicate that when the PRECONDITION is satisfied, the state of the cell changes to the designated POSTCONDITION, and its output is Delayed for the specified time. If model's state variables need to be modified, the ASSIGNMENTS section can be used (optional). The main operators available to define rules and delays include Boolean, comparison, arithmetic, neighborhood values, time, conditionals, angle conversion, pseudo-random numbers, error rounding and constants (i.e., gravitation, acceleration, light, Planck, etc.). At present, CD++ is able to execute models in single processor, parallel or real-time mode. The execution engine uses model's specifications, and it builds one object to control each component in the model hierarchy.

What is WebViewer?

A WebViewer is a Cell-DEVS Visualization and Analysis platform. Visualization and analysis of simulation results is normally conducted as a post-processing step, once a simulation has been successfully executed. The first step to analyze simulation results is often to simply visualize the output of the simulation. A proper visualization of the results provides a better understanding of the model's behavior and helps identify issues that were undetected in the modelling phase. Further analysis steps commonly include the use of visual statistical analysis tools such as charts and graphs to identify patterns and tendencies in the results. The Cell-DEVS WebViewer was originally developed by the Advanced Real-time Simulation Laboratory at Carleton University for academic use in courses and publications. It has since been subject to many improvements. It is developed entirely in HTML5 and JavaScript, requires no separate installation, can be run locally on a user's computer and minimizes external dependencies.

The Cell-DEVS WebViewer is a minimalistic client-side web application built entirely in HTML5 and JavaScript. It requires only two external dependencies, Whammy.js to record videos of canvas animations in .webm format and the Data-Driven Documents (D3) library for dynamic charts. In this section, the user interface, the data structures and the general flow of the application will be discussed. The WebViewer was originally built as an alternative to the desktop-only simulation viewer that was included in the CD++ development environment (Chidisiuc, Wainer 2007). It is meant to offer a better loading process for the simulation results, more control over the visualization and more user-friendly visualization of results.
The WebViewer has certain steps in order to visualize the simulation results, for example, Loading Simulation results and Simulation Playback.

A WebViewer is a web Application and web applications are always subject to changes for many reasons like improving user interactivity, introducing new features, to improve usability.

A current and latest version of WebViewer has been introduced to meet the user's requirement of visualizing the Classic DEVS simulation results. Visualization of simulation has a great importance, reading large log files and understanding the simulation results is a difficult file. Charts, Graphs derived from the large set of results can be easy to understand the behaviour of the simulation. Classic DEVS simulation results can be visualised using the following: a DEVS Diagram, representing the atomic, coupled and the input and output ports, a Model track chart (which is a line chart that represents the track of the models at a frame) and a state frequency chart(is a bar chart that shows the frequency of the different states, user can select the states). A DEVS diagram is designed using SVG (Scalable-Vector Graphics).

Error handling in JavaScript

Errors in JavaScript could get complex at certain times and when that happens, programmers have no one to blame but the themselves or the language itself. It would be great if we all knew how to detect these errors, expose them and prevent them from happening again.
Errors can be divided into different categories like, Syntax error, runtime error, logical errors, wrong inputs from the user, etc. and these errors needs to be gracefully handled by the developer and the user must know what went wrong when an error comes.
Runtime errors are called exceptions which happen during the execution: like, invalid user input, failed to connect to a database, system out of memory. All correctly written programs should deal with runtime errors and as long as exceptions handled, they don't necessarily indicate a bug or a serious problem. For instance, the runtime could throw a *file not found* exception, which might just mean that file has to be created first. When an exception happens, it should be handled properly, which means client should receive a friendly message, and it should be logged on a server so that you can always check the log for possible repetitive issues, which require improvement. This improves maintainability, extensibility, and readability.

Tooltip in JavaScript

The tooltip or info tip or a hint is a common graphical user interface element.

It is used in conjunction with a cursor, usually a pointer. The user hovers the pointer over an item, without clicking it, and a tooltip may appear—a small "hover box" with information about the item being hovered over. Tooltips do not usually appear on mobile operating systems, because there is no cursor.

Demonstrations of tooltip usage are prevalent on web pages. Many graphical web browsers display the title attribute of an HTML element as a tooltip when a user hovers the pointer over that element; in such a browser you should be able to hover over Wikipedia images and hyperlinks and see a tooltip appear. Some browsers, notably Microsoft's

Internet Explorer, will also display the alt attribute of an image as a tooltip in the same manner if an alt attribute is specified and a title attribute is not. This is an incorrect behavior of the alt attribute according to the W3C specification. If a title attribute is also specified, it will override the alt attribute for tooltip content.

CSS, HTML, and JavaScript allow web designers to create customized tooltip.

Scalable-Vector Graphics (SVG)

SVG is an XML language, similar to XHTML, which can be used to draw vector graphics, such as the ones shown to the right. It can be used to create an image either by specifying all the lines and shapes necessary, by modifying already existing raster images, or by a combination of both. The image and its components can also be transformed, composited together, or filtered to change their appearance completely.
SVG came about in 1999 after several competing formats had been submitted to the W3C and failed to be fully ratified. SVG is supported by all major browsers. A downside is loading SVG can be slow. SVG does offer benefits, some of which include having a DOM interface available for it, and not requiring third-party extensions. Whether or not to use it often depends on your specific use case.
HTML provides elements for defining headers, paragraphs, tables, and so on. In much the same way SVG provides elements for circles, rectangles, and simple and complex curves. A simple SVG document consists of nothing more than the <svg> root element and several basic shapes that build a graphic together. In addition there is the <g> element, which is used to group several basic shapes together. Starting from there, the SVG image can become arbitrarily complex. SVG supports gradients, rotations, filter effects, animations, interactivity with JavaScript, and so on. But all these extra features of the language rely on this relatively small set of elements to define the graphics area.

SVG is supported in all modern browsers and even a couple versions back in some cases.
Before starting you should have a basic understanding of XML or another markup language such as HTML. If you are not too familiar with XML, here are some guidelines to keep in mind:

- SVG elements and attributes should all be entered in the case shown here since XML is case-sensitive (unlike HTML).
- Attribute values in SVG must be placed inside quotes, even if they are numbers.

Implementation:

The following implementations were made to improve the user interactivity.

1. Error message should be displayed to a user in case anything goes wrong while visualizing the simulation. So, some of the common errors were handled and now the user would know if there is any issue while visualization. for example, error was not handled before when the user enters wrong files for the simulation.

User should enter the correct files and it should be mentioned to the user, the goal is to provide a proper message to the user in such cases, also we can provide indications to the user about what he should be doing, what happened, etc.

For Cell DEVS simulation: compulsory files to be uploaded for the simulation visualisation are .log and .ma file, other files like .pal file is optional. .pal defines the color scheme user wants to see in the simulation results.
For DEVS: a user needs to upload .log, .ma and .svg file to completely visualise all the simulation results that are in .log file.
There was a problem while saving the Dashboard, from 'Save Dashboard' option, which saves a JSON file for future purpose, as it can load the simulations directly with a JSON and log file. The Bug was fixed, and 'Save Dashboard' works fine now.

2. A user needs tutorial or help in many of the cases, like uploading files or when user needs to know how the playback widget works and what are the settings. These brief descriptions are provided to a user using a tooltip. Tooltips are text labels that appear when the user hovers over, focuses on, or touches an element. They may contain brief helper text about its function. For example, they may contain text information about actionable icons.
Tooltips, which is mostly displayed as a '?' symbol is added to the control the widget explaining the user how the control works. A tooltip is also added to the RISE list where a brief description is provided to the user about each model in a square box below the whole RISE list because it is difficult for a user to understand what a model is visualizing just from the given name of model in the list.

3. In the current version of the WebViewer, new updates were made to the WebViewer which enable the WebViewer to not just display Cell DEVS models but also the Classic DEVS models. It is the latest update to the WebViewer, and it needs some refining for better user experience. For example, In the current version, to visualize the DEVS Diagram, a user has to upload an SVG after loading the widget by selecting the widget name and loading it. This may not be convenient for the user, the SVG should be uploaded while uploading other files of the simulation results. These changes are implemented and now we are reading SVG along with other simulation logs and ma file, making it more convenient for the user. Now when a user uploads all the files, the results are loaded. User can now directly see the DEVS diagram results by loading DEVS diagram widget.

4. The current version of WebViewer displays the results of the simulation using a DEVS Diagram, State frequency chart and a model track chart but the current version only shows the results by highlighting the atomic and coupled models of a DEVS model. As we know that a DEVS model defines the atomic and coupled models as well as the input and output ports of each atomic and coupled components and also the link between the components, which is a crucial part of defining a DEVS model. So, the inputs and output ports are also meant to be read from the log file and highlighted for the perfect representation of simulation results.
   This project includes this reading of inputs and output ports from the logs of simulation results and mapping each of the component and ports to the .svg file, to correctly visualise the simulation results in DEVS diagram. Reading a log file is a challenge because it is a very large file that can take upto few GB's of memory, but the bowser has some memory limits. One of the HTML5's features, a FileReader object is defined that allows the browsers to read local files with byte-wise precision. We read the DEVS log files in smaller chunks of ~2MB each, processing each chunk into a more optimized internal data structure, then proceeding to the next chunk. This process repeated until the file has been read entirely.

Designing an SVG is a crucial part of the visualization and analysis for DEVS simulation results. A set of instructions needs to be provided to the user for perfect designing of SVG to display the results correctly. The SVG has been designed before but it needs to be properly documented for the users. User can use a third-party software to design SVG or write a code to design it. Some of the standard rules are defined below for designing SVG (Scalable-Vector Graphics). The next section of the paper focuses on the proper documentation of how the SVG files should be created by the user and what standards he requires to follow, so as to visualize the simulation correctly.

Standards Rules to define SVG file:

As we know, this software has been already designed, but it needs more clarifications, error handling and some standard rules with which user can define the SVG file tags in for proper display of results represented in the DEVS diagram.
The SVG were created for 2 different DEVS model using the standard rules, ABP (Alternating bit protocol) model and SARS (Student academic registration system) model. The ABP model was already existing in the system and was tested to display the new implementations of directly loading the SVG along with other files and the visualization of inputs and output ports implementation. So, some of the mandatory changes were made in SVG.
The Student academic registration DEVS model is a system that simulates the process of student applications. Student registration system keeps track of student's applications,

courses they have opted for and the status. Upon successful submission, a student ID is automatically assigned to the student for future reference purposes. Finally, within the registration period the Advisor do advising for students by approving/rejecting requested course by the student.

This model is also used as an example for creating SVG and testing it for the implementations done.

The software needs a proper documentation that can be provided to the users with the following set of results:

1. The file should be saved with a ".svg" extension
2. The SVG should be made as simple as possible by just using simple tags like <svg>, <path>, <rect> and <g> to not to create confusion to the user.
3. Every model and path from the DEVS model should be given an attribute 'model' in the SVG and the value of the attribute should be set to exact name of the model/path as specified by the user while creating a DEVS model.
4. All the extra or unused attributes should be removed, in case a third-party software is used for designing the SVG.

Images of the two examples of SVG images are available in Appendix A.

UI Improvements:

Using Color in an Educational/Research Website:

One of the most effective ways to boost learners' engagement and help them emotionally connect to the is to integrate color into the design. Color can give you the opportunity to transform any subject matter into a successful website even if it's educational, thanks to the fact that the human brain has the power to convert color into emotions and moods. A designer or a developer can consider the following facts while designing the color scheme.

1. Have an idea of the mood you're trying to achieve beforehand.
2. Don't go overboard when utilizing color in educational/research.
3. Never sacrifice legibility for aesthetic design.
4. Consider your learning audience's background.

So, Colors that can be used in the next deliverable can be:

- **Red**
  Symbolic of passion, energy, and excitement. If you want to guide your learners' attention to important information or want to boost their motivation, then you may want to go with red. This color can also be ideal for situations that call for immediacy. For example, if you want learners to ponder a particular topic or direct them to a specific section of the eLearning course or module at that very moment, using red can bring about a sense of urgency.

- **Orange**
  The color of optimism, communication, and mental stimulation. If you're dealing with a dull or complicated subject matter, use orange in order to make it seem more relatable and upbeat. Also, if you're trying

to boost learners' creativity, then you may want to consider adding orange into your eLearning deliverable's color palette.

- **Yellow**
  Symbolic of optimism, intellect, and cheerfulness. It can also help to boost memory and stimulate mental function. If you're trying to increase knowledge retention or make a potentially dull subject more exciting, opt for yellow. Be careful that it's a warmer shade, however, as bright yellow can be hard to read on the screen.

- **Green**
  The color of growth, balance, and tranquility. It's the ideal color to use when you're striving for a fresh design that is balanced. It is also known as the color of peace, which means that you may want to use green if you're trying to settle nervous learners before a big exam.

- **Blue**
  A favorite of corporate and many big brand logos, blue is seen as calm, trustworthy, and clear. Reflects feelings of peace, calm, and trust. It also happens to be the most favorable color. Blue is the go-to color for eLearning professionals who are looking for a way to transform a complex subject into one that is easily absorbed, given that it helps to make the subject matter seem less confusing and complicated.

- **Purple**
  Symbolic of imagination, fun, and sophistication. If you're trying to create a more upbeat learning atmosphere or one that encourages learners to have fun and get excited about the educational process, then purple is the ideal choice. opt for purples with blue undertones when striving for a relaxed mood, and more red undertones if you're trying to excite and engage learners.

- **Brown**
  This color brings about feelings of security and friendliness, while still conveying a sense of seriousness. If you're looking for a more neutral color that also helps to balance out the overall mood of the eLearning course, then you may want to go with brown.

To sum those up, cool colors like green, purple, and blue are calming and can help learners focus.

Using HCI Principles for CD WebViewer UI improvements:

Human-computer interaction (HCI) is a multidisciplinary field of study focusing on the design of computer technology and, in particular, the interaction between humans (the users) and computers. While initially concerned with computers, HCI has since expanded to cover almost all forms of information technology design.

Norman's 7 Principles:

1. Use both knowledge in the world and knowledge in the head.
2. Simplify the structure of tasks.

3. Make things visible: bridge the gulfs of Execution and Evaluation.
4. Get the mappings right.
5. Exploit the power of constraints, both natural and artificial.
6. Design for error.
7. When all else fails, standardize.

Object Oriented Programming Paradigm (OOPP)

The CD WebViewer uses the Object-Oriented Approach to meet user's requirement and make the coding or development simpler for the new users. The Object-Oriented programming paradigm plays an important role in **human computer interface**. It has different components that takes real world objects and performs actions on them, making live interactions between man and the machine. Following are the components of OOPP −

- This paradigm describes a real-life system where interactions are among real objects.
- It models applications as a group of related objects that interact with each other.
- The programming entity is modeled as a class that signifies the collection of related real-world objects.
- Programming starts with the concept of real-world objects and classes.
- Application is divided into numerous packages.
- A package is a collection of classes.
- A class is an encapsulated group of similar real-world objects.

HCI principles can be used in CD WebViewer to improve consistency and eliminate confusion. Providing user to provide information about how the dashboard and control works. Providing User more control and freedom, flexibility and efficiency to use. Minimize the number of inputs from the user is a part of a good UI design. Whenever possible, design systems so that potential errors are kept to a minimum. Error prevention is very important. Users do not like being called upon to detect and remedy problems, which may on occasion be beyond their level of expertise. Eliminating or flagging actions that may result in errors are two possible means of achieving error prevention.

Provide User help and documentation. Ideally, we want users to navigate the system without having to resort to documentation. However, depending on the type of solution, documentation may be necessary. When users require help, ensure it is easily located, specific to the task at hand and worded in a way that will guide them through the necessary steps towards a solution to the issue they are facing.

One of the ways to achieve a better user interaction in CD WebViewer is to divide the WebViewer into two parts, one for the Cell-DEVS and another for Classic DEVS models and Modify the widget configuration process.

Results:

The implementations described in the section above were implemented and the results were tested for 2 different DEVS model, ABP (Alternating bit protocol) model and SARS (Student academic registration system) model. The ABP model was already existing in the system and was tested to display the new implementations of directly loading the SVG along with other files and the visualization of inputs and output ports implementation.

The Student academic registration DEVS model is a system that simulates the process of student applications.

Following figures shows the results of the simulation as implemented on both ABP and SARS systems at different timesteps, which shows the inputs and the output ports highlighted along with the models, the models are filled with a different color whereas the links are highlighted by thickening the border/line or in terms of SVG, it's called increasing the stroke- width :

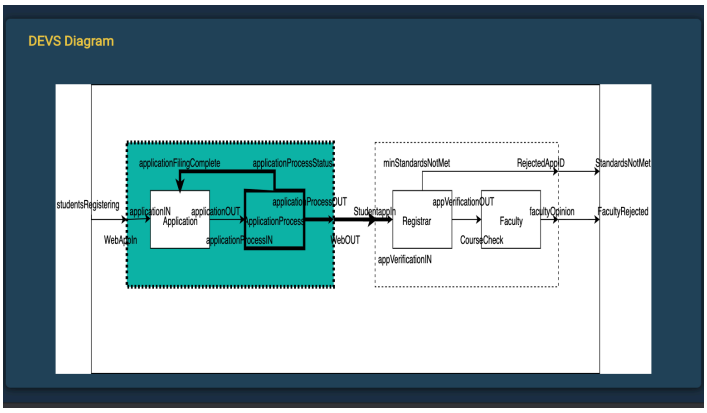

Figure 3 represents the simulation results through DEVS diagram for SARS model at time step 1.

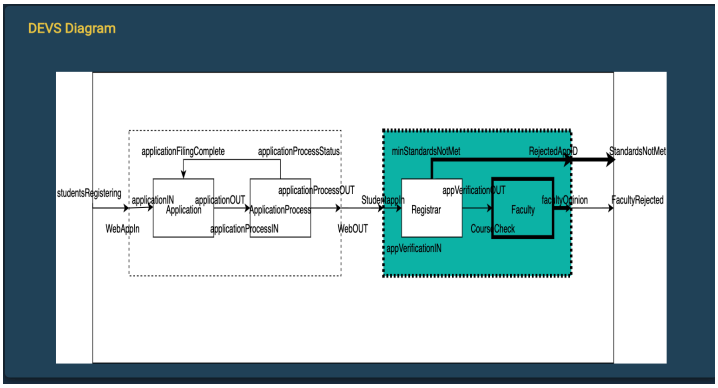

Figure 4 represents the simulation results through DEVS diagram for SARS model at time step 2.

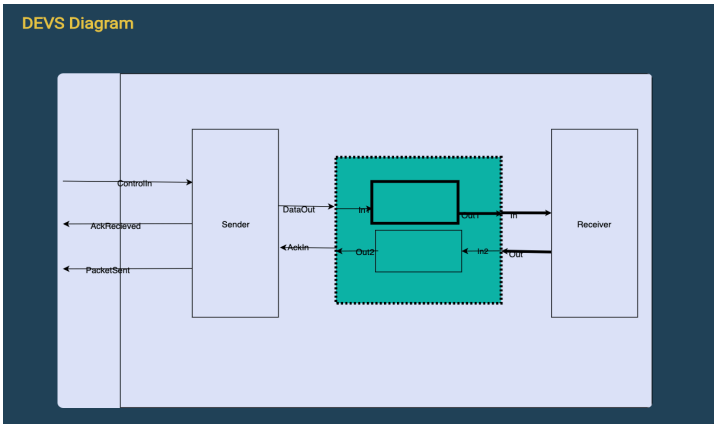

Figure 5 represents the simulation results through DEVS diagram for ABP model at time step 1.

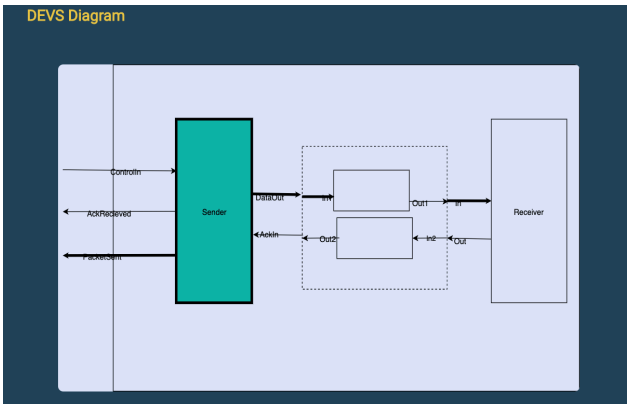

Figure 6 represents the simulation results through DEVS diagram for SARS model at time step 1.

The following figures shows the results of tooltips that were added to the rise list and the control widget:

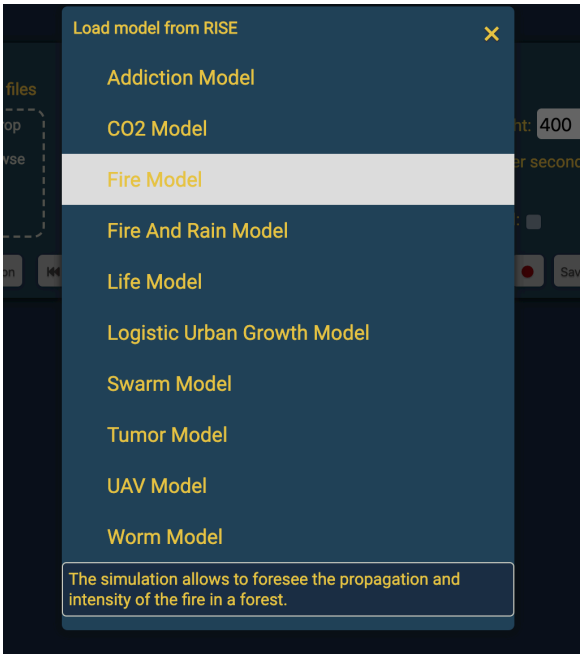

Figure 7 represents the rise list, when the mouse is over the selected model, it shows the description of the model.

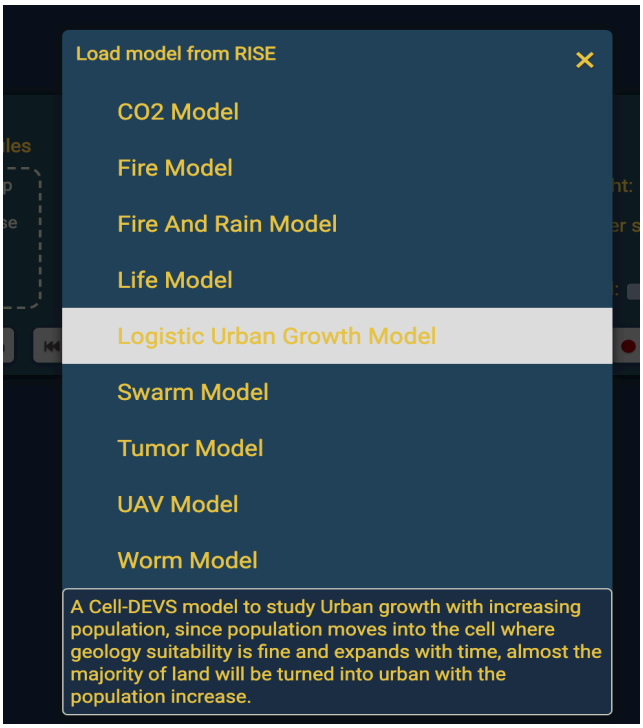

Figure 8 represents the rise list, when the mouse is over the selected model, it shows the description of another model.

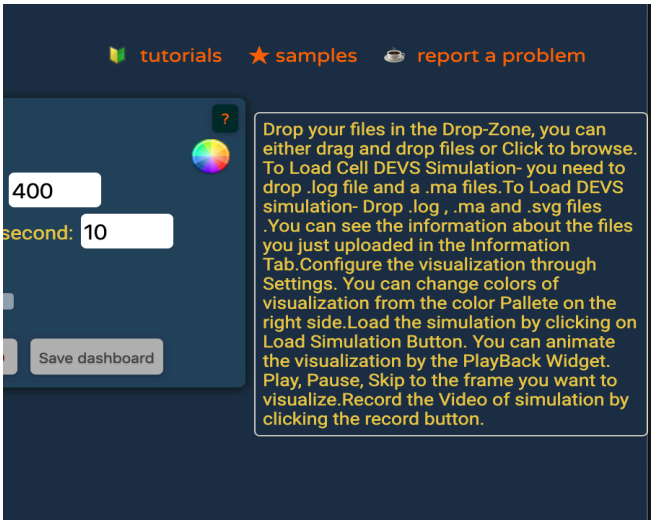

Figure 9 represents the new tooltip feature added on the control widget, to help user know how control works.

The following screenshot shows the alert when error was handled while user inputs the wrong file to the drop zone.

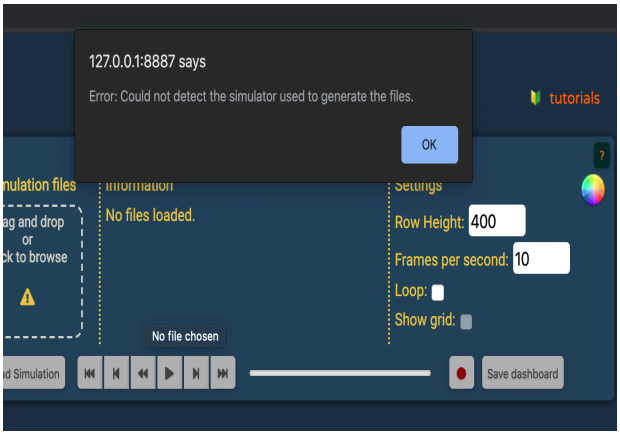

Figure 10 shows the error message alert user gets when wrong files are entered by the user.

The following image shows that the user can directly load the .svg by dropping the DEVS diagram along with .log and .ma files and then loading the DEVS diagram widget directly.

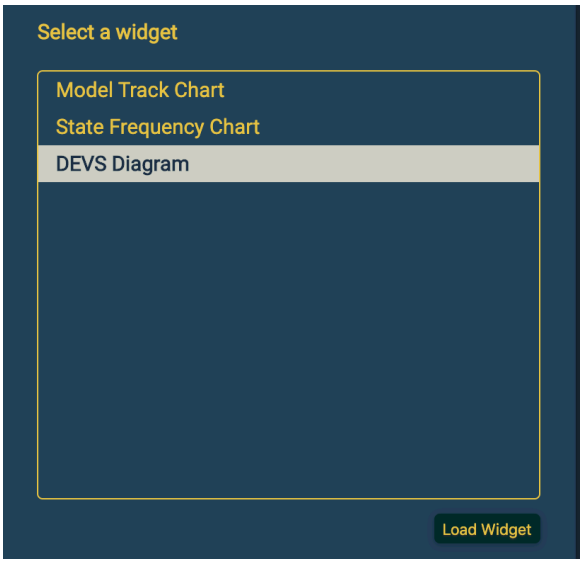

Figure 11 represents widget where user can now directly click on “Load Widget” and will be able to see the DEVS diagram.

Conclusions and Future Works:

In this paper, we conclude that the latest version of WebViewer is designed in order to visualize the simulation of Classic DEVS results and many improvements were made to the existing software to make it more consistent for the user and easy to use by providing tutorials, helps and show a proper error message in case something goes wrong.

Also, Code has been modified to visualize all the Classic DEVS results, which includes models, input ports and the output ports. Providing clear feedback to the users through interaction with diagram. The .svg file has been loaded to the WebViewer at the same time as the simulation files to improve consistency and more convenient for the users.

As the web applications are always a subject of improvements. Some kinds of improvements are always needed in a web application to meet user requirements, providing new features to the users, etc. and that is why many improvements can be made to the WebViewer like reading logs of Cadmium and then visualising the Cadmium results. Developing the improved User Interface using HCI principles. Adding more charts for the DEVS models, like we have for Cell DEVS models.

The source code of the project is available at the ARS lab repository’s branch: https://github.com/SimulationEverywhere/CD-WebViewer-2.0/tree/DEV-Shreya

Appendix A:

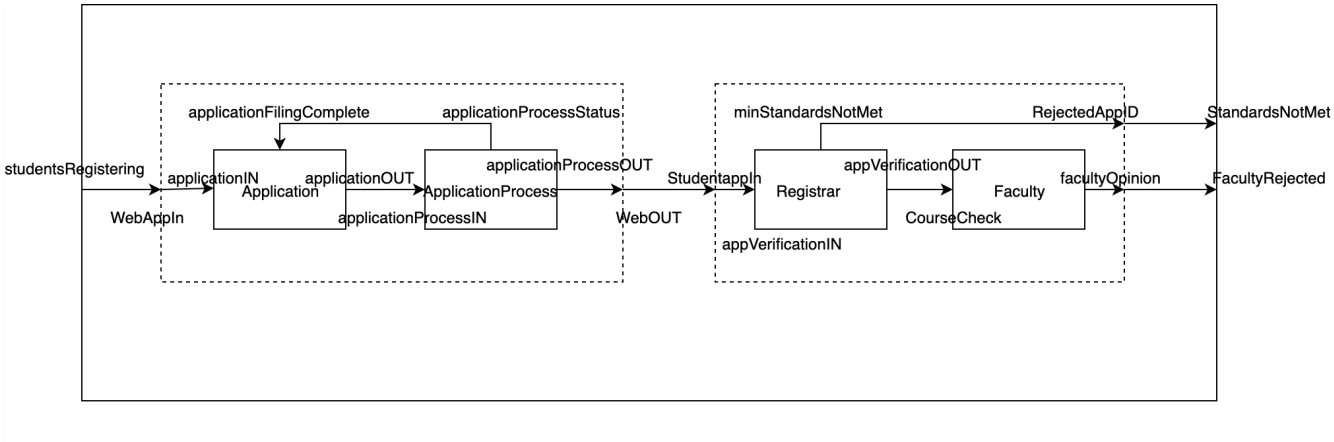

Figure 12 is the SVG for DEVS diagram for Student Academic Registration System used as an example to show results of simulation.

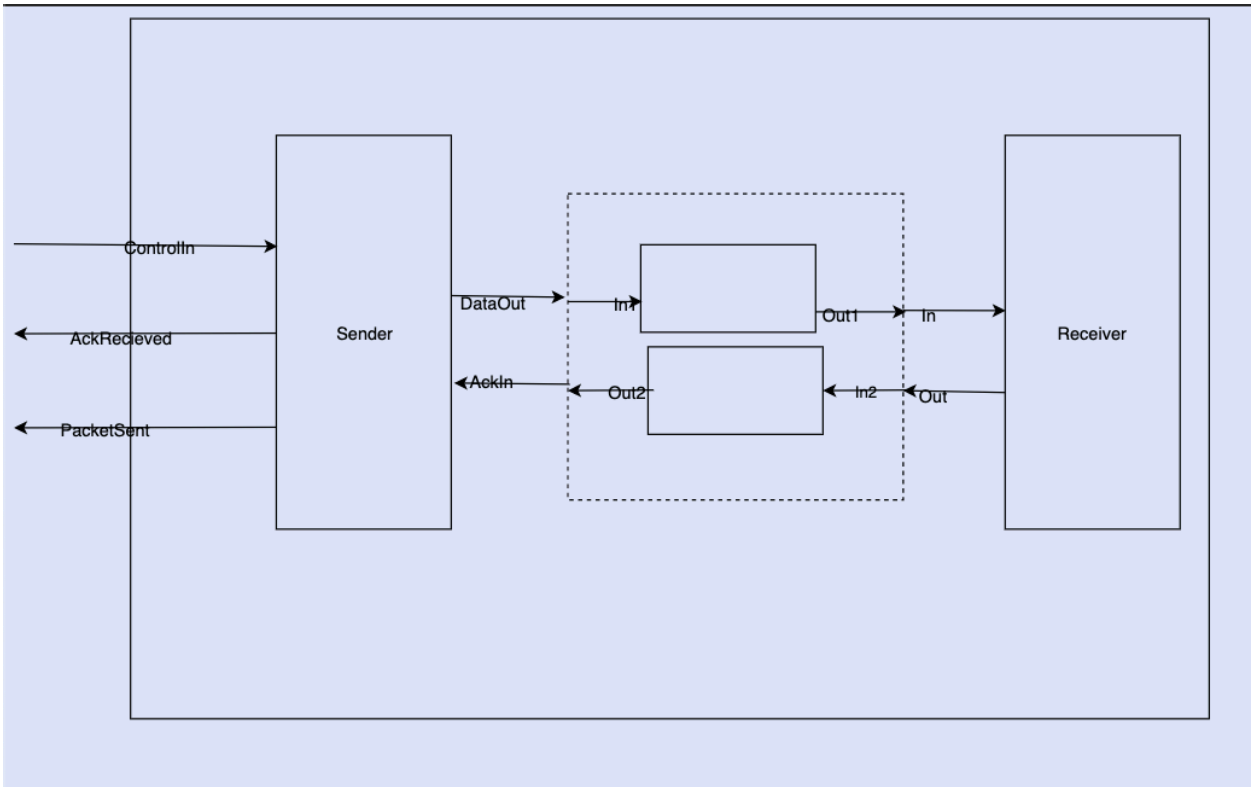

Figure 13 is the SVG for DEVS diagram for Alternate Bit Protocol used as an example to show results of simulation, that is already existing in the software.

References

1. https:// developer.mozilla.org/en-US/docs/Web/SVG/Tutorial/

2. Modeling and simulation of complex systems with Cell-DEVS. A Cell-DEVS model for logistic urban growth Bruno St-Aubin Gabriel Wainer

3. Analytics and visualization of spatial models as a service Bruno St-Aubin Eli Yammine Majed Nayef GabrielWainer

4. A CELL-DEVS model for logistic urban growth Bruno St-Aubin Gabriel Wainer

5. https://www.tutorialspoint.com/human_computer_interface/object_oriented_programming.htm

6. https://elearningindustry.com/4-tips-use-color-in-elearning

7. http://sce.carleton.ca/faculty/wainer/wbgraf/doku.php?id=model_samples:start