DEVS WebViewer Tutorial

Version 1.0

DRAFT

Advanced Real-Time Simulation Laboratory

October 12th, 2020

Carleton UNIVERSITY
Canada's Capital University

Contents

Carleton UNIVERSITY
Canada's Capital University

Introduction

The DEVS web viewer is a Web application developed in HTML and JavaScript. It relies on quickly maturing tools such as HTML5, the Canvas object, WebGL and the FileReader API. It provides an animated visualization of the simulation output for DEVS and Cell-DEVS simulation results. The application can convert original log files for multiple DEVS simulators into a common specification. In this way, it decouples the visualization from the simulator.

It uses SVG diagrams for DEVS models and a Canvas based representation for Cell-DEVS models. The simulators currently supported by the visualizer are CD++ (DEVS and Cell-DEVS), CD++2.0 (Cell-DEVS), and Cadmium (DEVS and Cell-DEVS). In this document, we will first provide a step-by-step tutorial to load both DEVS and Cell-DEVS results in the WebViewer. The second section will explain the various features of the graphic user interface. Following sections will delve deeper into the mechanics underlying the DEVS WebViewer.

Visualization of DEVS and Cell-DEVS results:

The WebViewer is a web application that will allow you to visualize DEVS and Cell-DEVS models online. The recommended browser to access the application is Chrome, Opera or Chromium-based browsers (some features may not work properly using other browsers). The application can be accessed at the following address:

http://ec2-3-235-245-192.compute-1.amazonaws.com:8080/devs-viewer/app-simple/

You will see this interface:

After that you will be able to visualize DEVS models or Cell-DEVS models as follows.

DEVS Results: Alternate Bit Protocol

In this section, we will demonstrate, step-by-step, how to load and visualize DEVS simulation results.

This procedure assumes that the simulation loaded is not in the DEVS WebViewer format and that a *diagram.svg* file is not provided.

> ** NOTE: Results can be converted manually using the procedure "Appendix 1 - Manual conversion of results to the DEVS WebViewer format" in the Appendix. In this case drop the converted results into the dashed box instead of the original simulation output files then, proceed normally.*

1. Convert the simulation output files to the DEVS WebViewer format. To do this, drag and drop the output files onto the dashed box in the GUI. Single or multiple files can be added. We have made available a complete example at:

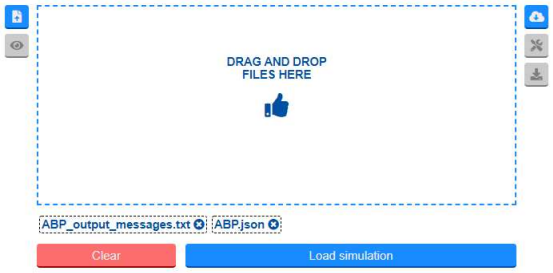
   http://www.sce.carleton.ca/courses/sysc-5104/lib/exe/fetch.php?media=apb-viewer.zip

   Sample input files can be found in folder ".\Sample Input\ABP - Cadmium - DEVS".

The folder contains 2 files:

a. ABP.json: A *json* file containing the ABP model information
b. ABP_output_messages.txt: A *txt* file containing all the output messages for the simulation, generated when you execute the APB model

2. The dropped files will appear below the dashed box:



~~2.~~3. At this point, you can:

- Remove a file by clicking on its name below the dashed box.
- Replace a file by first removing it then dropping the replacement file in the dashed box.
- Clear all files by clicking the "clear" button.
- Proceed by clicking the "Load Simulation" button

~~2.~~4. Click on the "Load Simulation" button, you will see a warning that the *diagram.svg* was not loaded properly. This is expected behavior. To avoid showing this message, a well-formed *diagram.svg* should be provided:

~~3.~~5. Click on the "Download normalized simulation files" button on the right-hand side of the application. This will download a zip file with the converted results. The zipped file should contain an *options.json*, *structure.json* and a *messages.log* files. The *messages.log* contains a normalized log file for the visualization tool. *Options.json* includes configuration information for the visualization parameters selected (this is useful for Cell-DEVS models). The *structure.json* contains the basic structure of the coupled model.

~~4.~~6. At this point, you must create a *diagram.svg* file. There are many ways to do this, you can use a desktop application such as Inkscape (https://inkscape.org/) or an online tool such as Diagrams.net (https://app.diagrams.net/). Once you have drawn and exported an *.svg* file, it may be convenient to format it to be more readable. SVG follows the HTML format so an HTML formatter tool can be used, for example, FreeFormatter (https://www.freeformatter.com/html-formatter.html).

~~5.~~7. For the viewer to animate the simulation's execution trace, structural elements of the model (atomic and coupled models, ports and couplings) must be associated to SVG elements. This is done by assigning an id to SVG elements and referencing that id in the *structure.json*. Any number and type of SVG elements can be associated to a structural element. We provide an example below for the "sender" atomic model in the ABP model:

*** A completed diagram file for this example can be found in the ".\Sample Output\ABP - Cadmium - DEVS" folder. The file is named diagram.svg.*

SVG Fragment for the Sender atomic model in *diagram.svg*

As can be seen below, multiple SVG elements can be used to represent structural components of a single model. In this case:

- The m-04 *rect* element will represent the "receiver" atomic model
- The t-04 *text* element will label the "receiver" atomic model
- The p-09 and p-10 *text* elements will represent ports of the "receiver" atomic model
- The l-08 *path* element will represent the coupling between the "receiver" atomic model and the "network" coupled model

```
<rect id="m-04" width="24.8" height="79.8" x="177.1" y="66.1/>
<text id="t-04" y="107.8" x="178.7">Receiver</text>
<text id="p-09" x="173.0" y="88.0">in</text>
<text id="p-10" x="170.1" y="120.7">out</text>
<path id="l-08" d="m 177.32701,122.44053 h -17.5" />
```

Associating node elements from the "nodes" array in *structure.json*.

As can be seen below, both the m-04 and t-04 SVG elements will be associated to the receiver element. In this case, t-04 is a text SVG element used to label the model and m-04 is a rect element.

```
    {
        "name": "receiver",
        "type": "atomic",
        "svg": ["#t-04", "#m-04"]
    }
```

Associating port elements from the "ports" array in *structure.json*.

As can be seen below, the "in" input port for the "receiver" atomic model will be associated to the p-09 SVG element. The "out" output port for the "receiver" atomic model will be associated to the p-10 SVG element. In this case, both p-09 and p-10 are text SVG elements used to visualize ports.

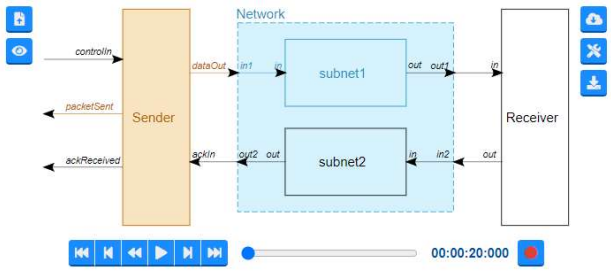
```
    {
        "model": "receiver",
        "name": "in",
        "type": "input",
        "svg": ["#p-09"]
    }, {
        "model": "receiver",
        "name": "out",
        "type": "output",
        "svg": ["#p-10"]
    }
```

Associating link elements from the "links" array in *structure.json*.

As can be seen below, the coupling between the "out" port on the "receiver" atomic model and the "in2" port of the "network" coupled model will be associated to the p-09 SVG element.

```
    {
        "modelA": "receiver",
        "portA": "out",
        "modelB": "network",
        "portB": "in2",
        "svg": ["#l-08"]
    }
```

5.8. Once this is complete, the simulation output files including the newly created *diagram.svg* file can be reloaded as specified in step 1. At this point, the diagram should be visible and ready to be animated as specified in the next section.


Network
controlIn
dataOut
in1
in
subnet1
out
out1
in2
Receiver
packetSent
Sender
ackReceived
ackIn
out2
out
subnet2
in
in2
out
00:00:20:000

Cell-DEVS Results: Logistic Urban Growth

In this section, we will demonstrate, step-by-step, how to load and visualize Cell-DEVS simulation results. This process assumes that the simulation loaded is not in the Cell-DEVS WebViewer format.

1. If the results are from the Cadmium simulator, you must first create a *.json* file (any name will do) containing the following parameters:

   - name: the name of the simulation model, LUG in this case
   - ports: a list of port names to assign to message values, in this case each message carries two values, the "type" of the cell and its "weight".

     Note: Cadmium does not use ports. Instead, output messages can contain multiple values. As a temporary workaround, each of these values is assigned to a port so that the data can be processed the same as for other simulators. Therefore, each value must be associated to a named port.
   - size: the size of the cell-space for the model, in this case, the size is 100 x 100 x 2.

   And here is an example of the completed *.json* file

```
{
   "name": "LUG",
   "ports": ["type", "weight"],
   "size": [100,100,2]
}
```

2. Convert the simulation output files to the DEVS WebViewer format. To do this, drag and drop the output files onto the dashed box in the GUI. Single or multiple files can be added.

   ** NOTE: Results can be converted manually using the procedure "Appendix 1 - Manual conversion of results to the DEVS WebViewer format" in the appendix 1. In this case drop the converted results into the dashed box instead of the original simulation output files then, proceed normally.*

   We have made available a complete example at:
   http://www.sce.carleton.ca/courses/sysc-5104/lib/exe/fetch.php?media=apb-viewer.zip

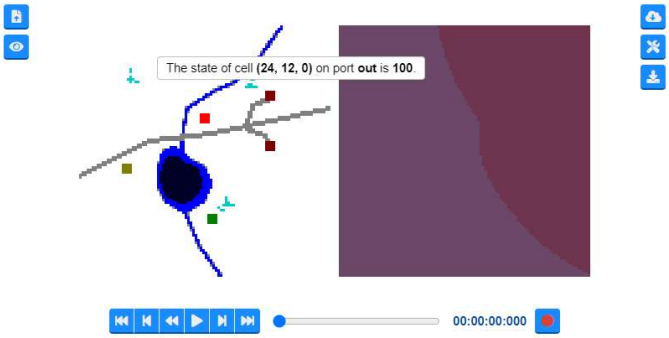
Commented [BS1]: update

3. The dropped files will appear below the dashed box:

~~5.~~9. At this point, you can:

- Remove a file by clicking on its name below the dashed box.
- Replace a file by first removing it then dropping the replacement file in the dashed box.
- Clear all files by clicking the "clear" button.
- Proceed by clicking the "Load Simulation" button

4. Click on the "Load Simulation" button. At this point, if a *.pal* file was provided, you will see the simulation with the proper color scheme:



If a "pal" file was not provided, then a completely black cell-space will show rather than the proper color scheme:

5. In both cases, the color scheme of the visualization can be modified through the settings widgets, as described in the following section.

DEVS WebViewer user interface and features:

The figure shows the user interface presented to the user when the DEVS Web Viewer is accessed. The user can click on the central drop zone to upload the files to convert or view, or they can drag and drop them directly into the drop zone. Users can upload the files to be converted as presented in section 1, or the common specification files as presented in section 2. For the latter, the files will be loaded directly in the application. For the former case, the files will be first converted then loaded in the application. The system automatically determines the simulator and formalism used.

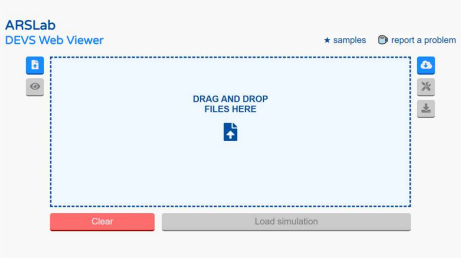


Figure 1: DEVS viewer initial display

The following figure shows the files ready to be loaded in the viewer. The file list can be emptied by clicking the “Clear” button or individual files can be removed by clicking the box with the corresponding file below the main input box. To load and visualize the simulation, users click the "Load simulation" button. At this point, the input files will be converted if required, then parsed and loaded. If the format of the provided files is adequate, the viewer will show the diagram if the analyzed model uses the regular DEVS formalism, or a grid if it follows the Cell-DEVS formalism.

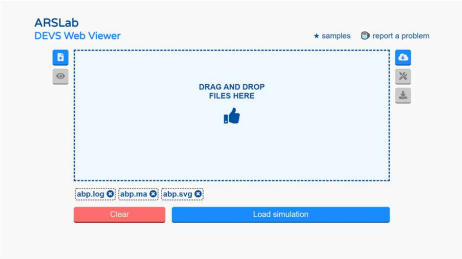


Figure 2: simulation files ready to be loaded in the viewer

There is a toolbar to the right of the file input box. The first button in the toolbar (cloud icon) allows users to load simulation results from the RISE platform. The RISE platform holds a collection of simulation results to use as demos. Below is a screenshot with some models currently available.

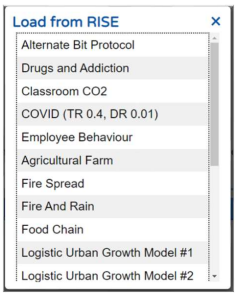


Figure 3: list with Models stored in RISE.

The second button, with a screwdriver and wrench icon, allows users to configure their visualization. This is only accessible once the simulation has been successfully loaded. Playback speed, layout, grid colors and diagram size can be configured. The figure below shows, on the left-hand side (a), the base configuration interface and, on the right-hand side (b), the grid configuration interface for Cell-DEVS models. The grid configuration allows users to specify which layers and which ports to show in the visualization as well as the colors used to draw cells.

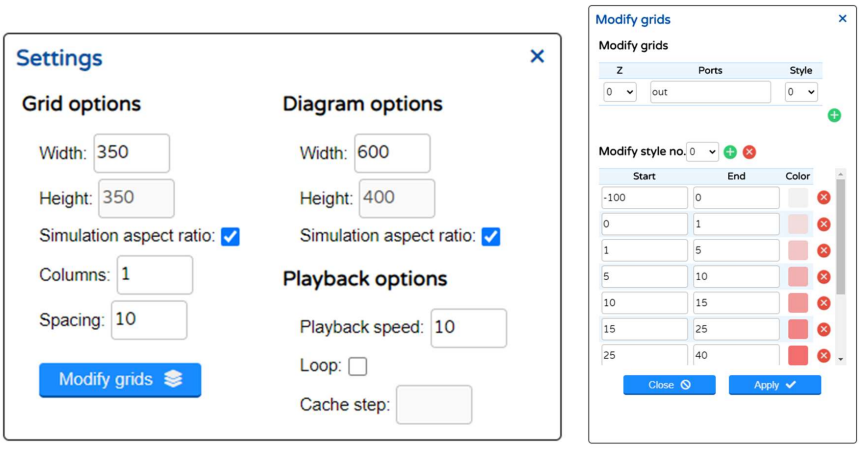


Figure 4: (a) Simulation settings. (b) Grid and style options.

The third button, with a download arrow icon, allows users to download the files in the common specification presented in section 2. This is only accessible once the simulation has been successfully loaded.

Finally, there is also a playback bar that allows users to navigate through the simulation time steps. The playback bar is located below the main simulation visualization. Users can move forward or backwards a single frame, animate the simulation backwards or forwards or jump

to the end or the beginning of the simulation. Users can also use the slider to move through the time steps of the simulation.

To the right of the bar, a record button allows users to record their simulation as a *.webm* video. To do so the user must click the record button then use the playback options to animate the simulation (all navigation options can be used) and finally, click the record button again to stop and download the video.

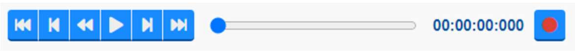


Figure 5: The simulation playback bar.

Conversion of simulation output to the common specification

The viewer can convert simulation outputs from DEVS simulators into a web optimized common specification. The files required vary according to the simulator and formalism used. In this section, we go over the process of converting the simulation outputs to the common specification.

Simulation structure

The conversion process requires information on the structure of the models that were simulated. The structure comprises atomic and coupled models, output ports and couplings between models. This information is used to optimize the log file, to traverse the model graph for visualization, to provide contextual information to users when they interact with the visualization, etc.

CD++ Simulators

The structure information is available in the *.ma* file. For regular DEVS models, the *.ma* file should contain the following elements. Some of them are optional:

- Models (required): each model is identified by a name within square brackets. In the example below, 2 models are identified (top and sender).
- Components (optional): each key-value pair is a submodel. A model with components is a coupled model. In the example below, the top model is coupled.
- Links (required): they correspond to the External Input Couplings, External Output Couplings, and Internal Couplings in the DEVS formal specification of the coupled models. In the example below, the top model has 7 links.

```
[top]
components : sender@Sender
components : Network
components : receiver@Receiver
out : packetSent ackReceived
in : controlIn

Link : controlIn controlIn@sender
Link : dataOut@sender in1@Network
Link : out1@Network in@receiver
Link : out@receiver in2@Network
Link : out2@Network ackIn@sender
Link : packetSent@sender packetSent
Link : ackReceived@sender ackReceived

[sender]
preparation : 00:00:10:000
timeout : 00:00:20:000
```

Figure 6: A partial .ma *file for the Alternate Bit Protocol, a CD++ DEVS model.*

For Cell-DEVS models, the *.ma* file should contain the following elements in addition to the ones listed above. Some of them are optional:

- Ports (optional): Ports are identified by the "neighborports" key. The value for this key-value pair is a list of named ports for the model. These will become available as layers in the visualization. The example below has no ports defined, only the "out" port is available.
- Dimensions (required): Dimensions are usually specified through the "dim" key but a combination of "width and "height" is also supported. These are the dimensions of the cell-space. In the example below, the dimensions are 100, 100, 2
- Initial Value (optional): The initial value used for the cell-space. These can be specified through the "initialvalue" or "initialrowvalue" keys. The *.val* file often supersedes these elements. In the example below, the initial global value is 0.

```
[top]
components : LUG

[LUG]
type : cell
dim : (100, 100, 2)
delay : transport
defaultDelayTime : 100
border : wrapped
neighbors : LUG(-1,-1,0) LUG(-1,0,0) LUG(-1,1,0)
neighbors : LUG(0,-1,0)  LUG(0,0,0) LUG(0,0,1) LUG(0,1,0)
neighbors : LUG(1,-1,0)  LUG(1,0,0)  LUG(1,1,0)
initialvalue : 0
initialcellsvalue : map.val
localtransition : LUG-rule
```

Figure 7: A partial .ma *file for the Logistic Urban Growth, a CD++ Cell-DEVS model.*

Cadmium simulator

In Cadmium, the structure information is output by the simulator alongside the simulation results when using the provided logger class. At the time of writing this document, this process has not been fully implemented with the Cell-DEVS version of Cadmium. Therefore, users need to write the *json* file discussed in this section manually. This file is based on the scenario files provided to the automated model generation process for Cadmium. An example is provided below:

```
{
    "scenario": {
        "default_cell_type": "CO2_cell",
        "default_state": {
            "counter": -1,
            "concentration": 500,
            "type": -100
        },
        "shape": [110,72,10]
   }
}
```

Figure 8: A sample json file that must be provided alongside the Cadmium simulation results for conversion

Here,

- default_cell_type is used to name the model; in this case CO2_cell
- default_state are the names of the fields in this object will be used to name ports
- shape is an array of integers contains the size of the cell-space; in this case is a 3D model of 100x72x10 cells.

Simulation results file

For CD++, the outputs are stored in a log file. This file always can be anything, but normally the examples you will find in our repositories use the *.log* extension for CD++, and the *.log01* extension for CD++2.0 (Cell-DEVS). In the case of Cadmium, there can be multiple log files, so the user has to choose the log file that contains the simulation message, specified with the "logger_messages" option in the main file where the top model is declared. Every user can choose a different option.

Initial values files

For CD++, Cell-DEVS models will often rely on an initial values file. When that is the case, the *.val* file must be provided. CD++2.0 (Cell-DEVS), does not require the initial values file since they appear in the log file at the first-time step.

Palette file

For CD++, users often specify style colors to draw the cell-space. For the conversion process to consider it, users must provide the *.pal* file.

For Cadmium Cell-DEVS models, users should provide a *style.json* that follows the specification detailed in section 2.

Diagram file

For DEVS models, regardless of the simulator used, users should provide a file that contains the diagram for the DEVS model. This should be an *.svg* file that follows the specification detailed in section 2.

Overview of the common specification:

The DEVS WebViewer requires that simulation results be provided in a specifically designed format. This format is obtained by converting simulation results using the process described in the previous section. The section describes this format, a summary data model is provided below:

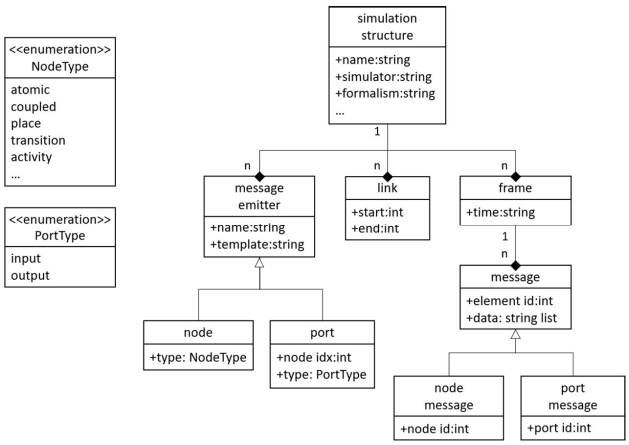


Figure 9: An overview of the common specification data model

The data model presented above contains a sub-structure to hold structural elements, message emitters and links, and another sub-structure to hold messages organized by time frames. In the implementation we have adopted, a *json* file is used to represent the former and a csv like format is used for the latter. Both are explained in this section.

structure.json

The *structure.json* file contains all information related to the structural elements of a DEVS or Cell-DEVS model (atomic and coupled models, ports, and links). The *messages.log* file contains all the messages output by the simulation. The figure below provides an example for the alternate bit protocol model (ABP):

```
{
   "name": "Alternate Bit Protocol",
   "simulator": "CDpp",
   "type": "DEVS",
   "nodes": [{
       "name": "sender",
       "type": "atomic",
       "svg": ["#m-01"]
   }, {
       "name": "receiver",
       "type": "atomic",
```

```
...
00:00:20:000;7,11;9,1
00:00:22:987;12,11
00:00:32:987;5,1
00:00:50:000;7,11;9,1
00:00:51:957;12,11
00:01:01:957;5,1
00:01:04:992;14,1;10,1
00:01:14:992;7,20;9,2
00:01:17:174;12,20
00:01:27:174;5,0
```

```
        "svg": ["#m-02"]
    },
    ...
    ],
    "ports": [{
        "model": "network",
        "name": "in1",
        "type": "input",
        "svg": ["#p-05"]
    }, {
        "model": "network",
        "name": "out1",
        "type": "output",
        "svg": ["#p-06"]
    },
    ...
    ],
    "links": [{
        "modelA": "network",
        "portA": "out1",
        "modelB": "receiver",
        "portB": "in",
        "svg": ["#l-06"]
    }, {
        "modelA": "network",
        "portA": "out2",
        "modelB": "sender",
        "portB": "ackin",
        "svg": ["#l-04"]
    },
    ...
    ]
}
```

```
00:01:44:992;7,20;9,2
00:01:48:841;12,20
00:01:58:841;5,0
00:02:02:496;14,0;10,0
00:02:12:496;7,31;9,3
00:02:15:942;12,31
00:02:25:942;5,1
...
```

Figure 10 example of a JSON structure file for the ABP model (left) and the corresponding messages file (right).

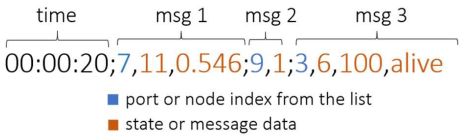
The *structure.json* file contains the following elements:

- name: The name of the simulation model
- simulator: The name of the simulator used
- type: The type of simulation model (DEVS or CellDEVS).
- nodes: an array of nodes representing the atomic and coupled models that compose the simulation model. Each model (atomic or coupled) contains the following elements:
    - name: the name of the model
    - type: the type of the model (atomic or coupled)
    - svg: the svg elements in the diagram that correspond to the model
    - size: (Cell-DEVS models only) an array of integer representing the dimensions of the cell-space for the model

- ports: an array containing ports that compose the simulation model. Each port contains the following elements:
  - model: the name of the model associated to the port
  - name: the name of the port
  - type: the type of the port (input or output)
  - svg: the svg elements in the diagram that correspond to the model
- links: an array containing links that relate different elements of the simulation model. Each link contains the following elements:
  - modelA: the name of the origin model for the link
  - portA: the name of the origin port for the link
  - modelB: the name for the destination model for the link
  - portB: the name of the destination port for the link
  - svg: the svg elements in the diagram that correspond to the model

messages.log:

All messages output by a simulation are contained in the messages.log file. Each line of the file contains all messages emitted for a given time step. Messages are constructed as specified below:



Messages differ slightly for Cell-DEVS models. For each state or message data, the first three values will correspond to the X, Y and Z coordinates of the cell. The next value will correspond to the port index in the structure and the remaining values will correspond to the message emitted.

diagram.svg

A scalable vector graphics (SVG) file is required to visualize DEVS models. The SVG file must show a diagram with the structure of the model that is going to be animated. It can be drawn with any SVG editing tool, such as Inkscape or diagrams.net. The visualizer gives a high degree of freedom in the design of the diagram, so the user can add any extra images or text that clarifies the content of the diagram. Here are recommended elements that should be in the diagram:

- Shapes for the atomic and coupled models included (can be rectangles, circles, or more complex shapes).

- A label with the name of each atomic or coupled model (can be inside or outside the model).
- Links composed by lines and arrow markers to show the direction of the link.
- Labels with the names of the input and output port of the models (the user could, eventually, obviate some of the ports that are not necessary for visualization, and still reproduce the visualization).

Each element in the diagram should have a unique ID that corresponds to an element in the *structure.json* file. The DEVS WebViewer requires this to highlight the active components in every time step.

Appendix 1 - Manual conversion of results to the DEVS WebViewer format

This procedure replaces step 1 of the "DEVS Results: Alternate Bit Protocol" procedure and step 2 of the "Cell-DEVS Results: Logistic Urban Growth" procedure. It is provided as an alternative to the web-based file conversion process. The 2 next procedures described in this section indicate when this procedure can be used as a substitute.

At the time of writing this document, this is the preferred method of conversion.

1. Download the converter application here : https://github.com/staubibr/arslab-logs/raw/master/sim.converter.jar.

   *** The application requires a java JRE or JDK 8 or above. This may already be installed on your system. If not, we provide an installation procedure for Windows 10 in appendix 2*

2. Open a command prompt or a terminal, navigate to the folder where sim.converter.jar is located.

3. Type in the following command:

   ```
   java -jar sim.converter.jar "[input]" "[output]"
   ```

   Where [input] is the input folder containing the simulation files and [output] is the output folder where the converted files will be saved. Paths can be absolute or relative to the current folder but they must point to an existing folder.

   We provide an example for both Cadmium DEVS and Cadmium Cell-DEVS. For both examples, the ".\output" folder should exist:

   a. For the Cadmium DEVS ABP results available in the ".\Sample Input\ABP - Cadmium - DEVS" folder. The following command will work:

      ```
      java -jar sim.converter.jar ".\\Sample Input\\ABP - Cadmium - DEVS\\" ".\\output\\"
      ```

   b. For the Cadmium Cell-DEVS CO2 results available in the ".\Sample Input\ - Cadmium – DEVS" folder. The following command will work:

      ```
      java -jar sim.converter.jar ".\\Sample Input\\CO2 - Cadmium - Cell-DEVS\\" ".\\output\\"
      ```

   The input folder provided to the converter application must contain the appropriate simulation files. The following table describes the files required by simulator:

Cadmium DEVS	▪ a *.json* file containing model information generated by the simulator ▪ a *.txt* file containing simulation messages generated by the simulator
Cadmium Cell-DEVS	▪ a *.json* file provided manually, see step 1 of the "Cell-DEVS Results: Logistic Urban Growth" procedure ▪ a *.txt* file containing simulation messages generated by the simulator
CD++ DEVS	▪ the *.ma* file required to execute the simulation ▪ the *.log* file output by the simulator
CD++ Cell-DEVS	▪ the *.ma* file required to execute the simulation ▪ the *.val* file required to execute the simulation (optional) ▪ the *.pal* file required to execute the simulation (optional) ▪ the *.log* file output by the simulator

4. If all goes well, you should see this in the command prompt or terminal:

```
reading source files...
converting source files to new specification...
converting palette if provided...
zipping everything...
writing to destination folder...
finished successfully.
```

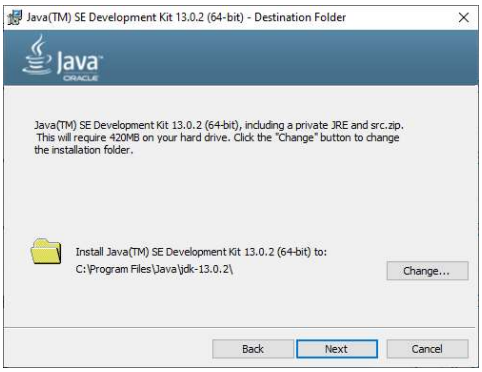
5. The converted simulation files will be in a zip archive at the specified output location.

Appendix 2 – Installing the required JDK (Windows 10)

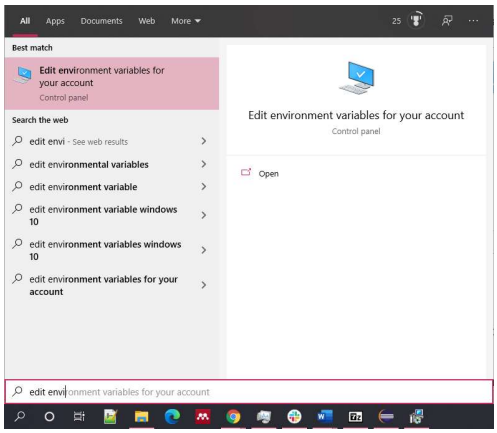
1. Download and launch the JDK 13 Windows x64 Installer. It can be found at the following URL:

   https://www.oracle.com/java/technologies/javase-jdk13-downloads.html

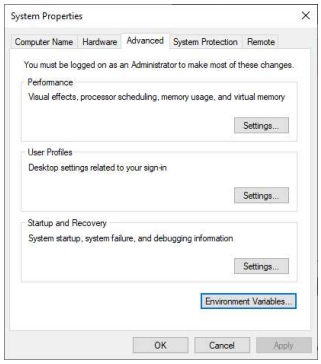
2. The installation is straightforward but note the install path from this window:



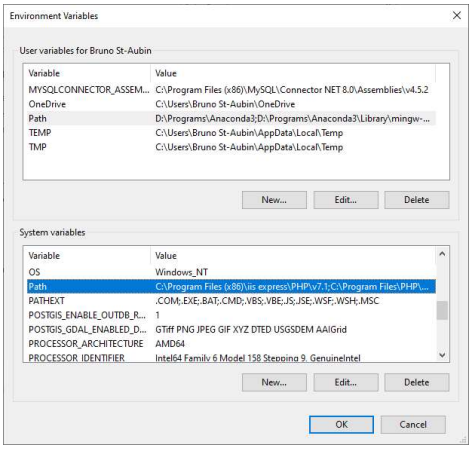
3. In the Windows search bar, type "Edit environment variables for your account". Click on the result as shown in the picture below:



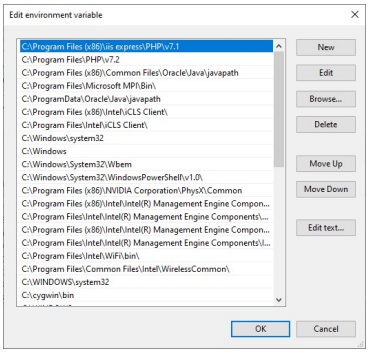
4. Click on the "environment variables..." button:



5. In the "Environment variables" window, click on the "Path" item from the "System variables" bottom list. Click on the "edit…" button.



6. From the "Edit environment variable" window, click on the "New" button:

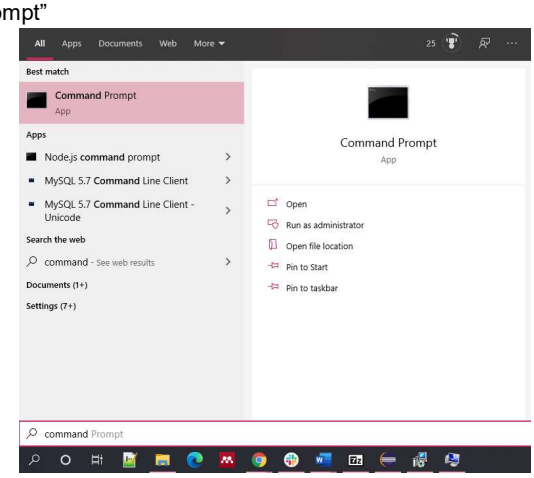


7. Type in the JDK install path noted in step b. Add bin after the last backslash. If you selected the default install path, your path should be similar or the same as the following:

```
C:\Program Files\Java\jdk-13.0.1\bin
```

8. Move the newly created path to the top of the list by first clicking on it, then clicking on the "Move Up" button until it is at the top of the list.

9. Confirm that the new version of Java is correctly installed. In the Windows search bar, type "Command prompt"



10. In the window that opened, type "java -version". If you see the following, the installation was successful.