# DEVS WebViewer tutorial

## 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 briefly explain the process to convert simulation log files to the common specification. Then we will provide an overview of the common specification format. Finally, we will explain the user interface and the different features of the viewer.

## 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 1: 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 2: A partial .ma file for the Logistic Urban Growth, a CD++ Cell-DEVS model.

#### Cadmium simulator

The structure information is output by the simulator alongside the simulation results when using the provide 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 may have to write this *json* file manually. This file is based on the scenario files provided to the automated model generation process for Cadmium. Therefore, a summary explanation may be useful:

* default\_cell\_type: the value will be used to name the model
* default\_state: the names of the fields in this object will be used to name ports
* shape: this array of integers contains the size of the cell-space

An example is provided below:

{

"scenario": {

"default\_cell\_type": "CO2\_cell",

"default\_state": {

"counter": -1,

"concentration": 500,

"type": -100

},

"shape": [110,72,10]

}

}

Figure 3: A sample json file that must be provided alongside  
 the Cadmium simulation results for conversion

### Simulation results file

For CD++ the outputs are stored in a log file. This file always has the *.log* extension for CD++ but sometimes 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.

### 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 4: 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",  "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"]  },  ...  ]  } | ...  00:00:20:000;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  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 5 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:

Logo

Description automatically generated

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.

## 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.

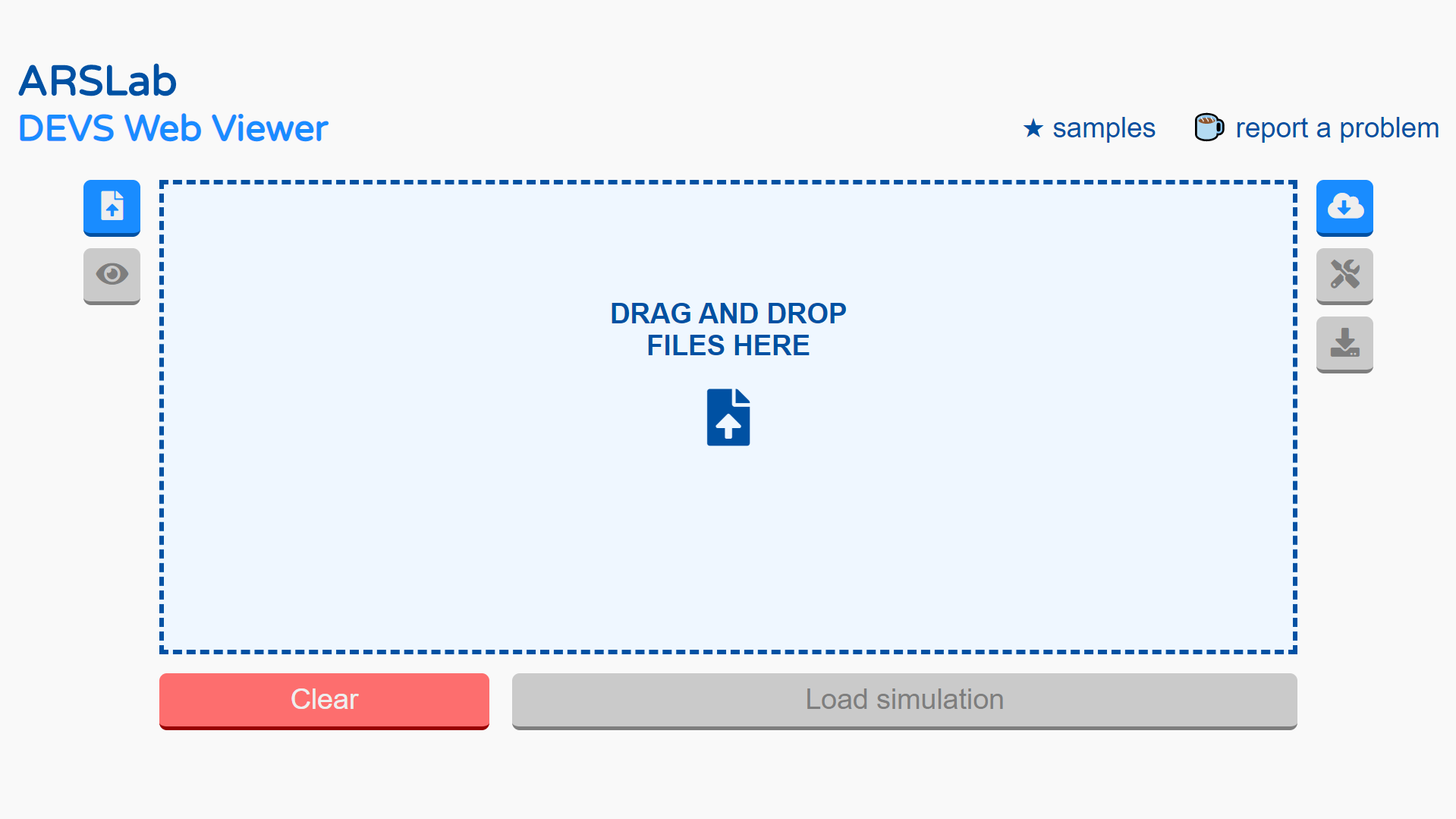


Figure 6: DEVS viewer initial display

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.

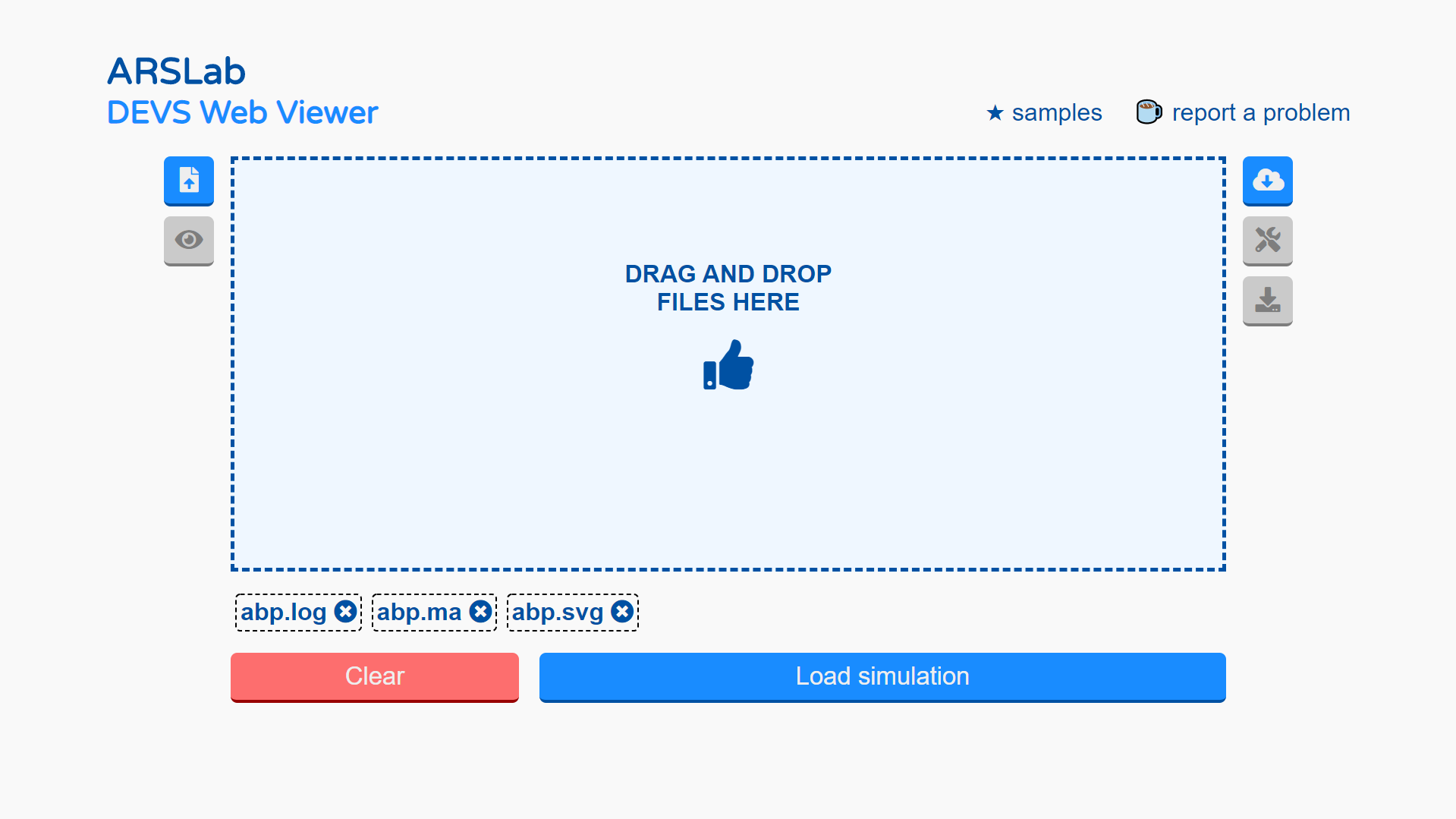


Figure 7: simulation files ready to be loaded in the viewer

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.

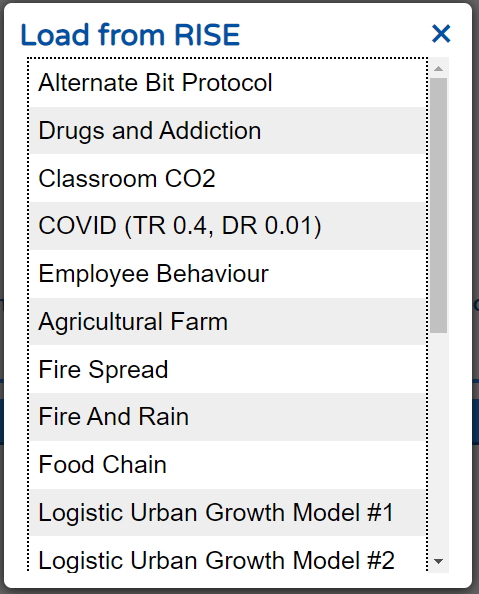


Figure 8: list with Models stored in RISE.

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 colours 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 colours used to draw cells.

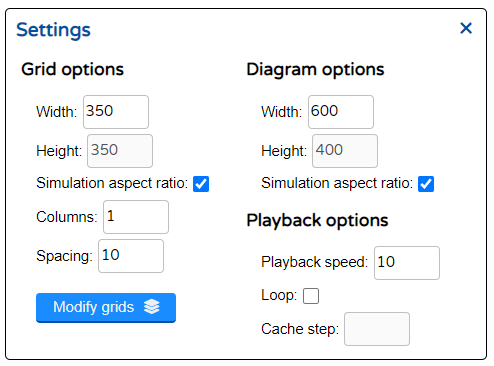
 

Figure 9: (a) Simulation settings. (b) Grid and style options.

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 10: The simulation playback bar.

## Examples

### DEVS visualization - The Alternate Bit Protocol

As an example, for DEVS visualization we present the Alternate Bit Protocol (ABP) model, shown in the figure below. The top coupled model is composed of the sender and receiver atomic models, and a network coupled model with two subnet atomic models. The model implements a simple protocol to achieve reliable communication in an unreliable network.

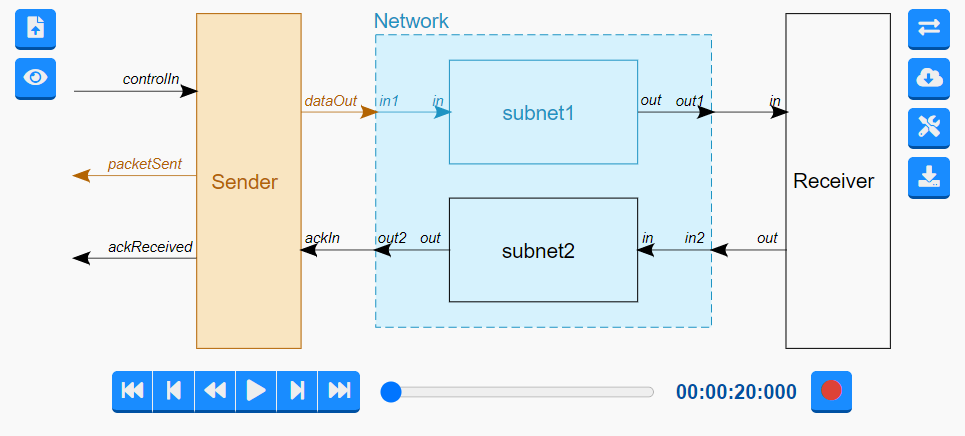


Figure 11: ABP model visualization

1. Prepare the files

As described in the previous section, to visualize the model, we need to provide the application with the necessary files. For both cases, the first step is to create the diagram.svg file for the model.

* Any SVG shape can be used for the structural components of the model. This includes models, both atomic and coupled, input and output ports, and couplings.
* Each shape associated with a structural element should have a valid id attribute. They can be assigned with any SVG drawing tool (i.e inkscape), or by manually editing the SVG in a text editor.
* Labels can be provided for any structural element using text SVG elements. They should also have an id attribute.
* Any elements that are not present in the SVG will not be highlighted during the visualization.

When the SVG file is ready, the second step is to link the structure.json file with the diagram.svg file. This is done by defining and assigning the ”svg” property on each structural element of the structure.json file. The “svg” property of each model, port or link in structure.json should be equal to a list of strings that contains the ids of the corresponding SVG shapes in the diagram.svg file.

1. Drag the simulation log (\*.log), the diagram (\*.svg) and structure (structure.json) files and load them by pressing the “Load simulation” button.
2. The diagram should be displayed. Review the visualization settings to change the size and playback speed if needed.
3. Use the playback buttons to navigate the simulation visualization. The models that are sending messages are highlighted in orange, and the models that are receiving messages are highlighted in blue, Hovering the mouse on top of the orange models shows the messages that are being output.
4. (Optional) Download the files with the button in the right bottom of the viewer.

### Cell-DEVS visualization - Classroom CO2

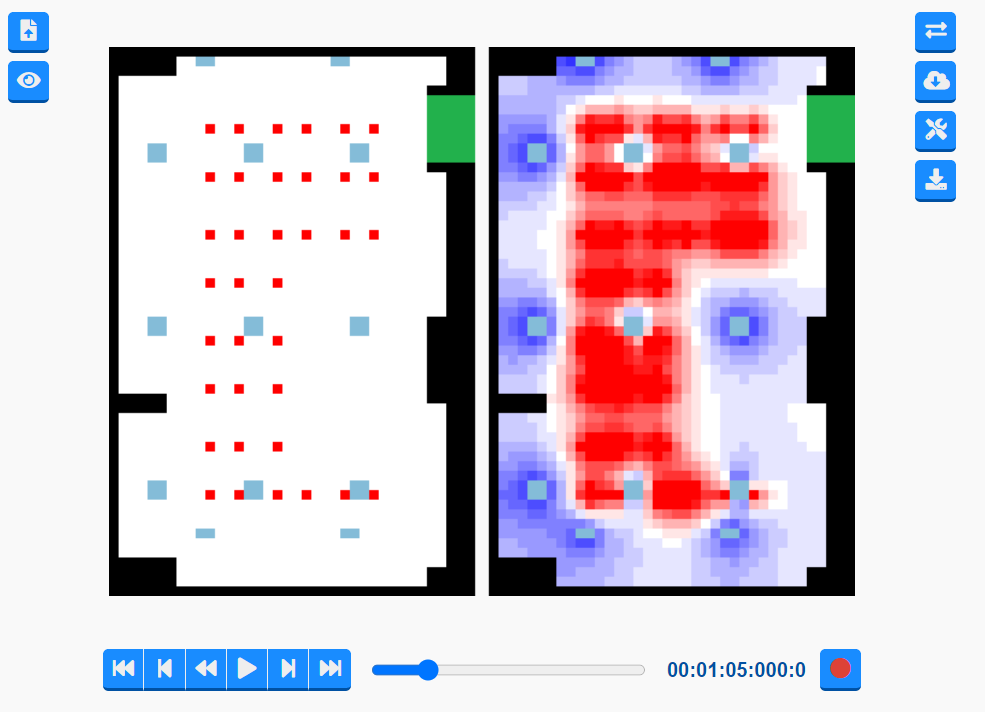


Figure 12: visualization for the CO2 concentration model

1. Prepare the files by converting original output files as specified in section 1.
2. Drag the simulation log (\*.log), structure (structure.json) and the style (style.json) files and load them by pressing the “Load simulation” button.
3. The diagram should be displayed. Review the visualization settings to change the number of columns and size of the grids, and the style for each port and grid.
4. (Optional) Download the files with the button in the right bottom of the viewer.