FPP Graphical Editor Developer's Guide

The Jet Trio (Team NASA JPL)
Wei-Hsuan Wang, Daiyi Lyu, Minghui Tang

Overview

FPP Graphical Editor is an Electron[1]-Vue[2]-Typescript[3] project based on the Electron-Vue skeleton. Electron is an open-source project based on Node.js and Webkit for building cross-platform desktop apps with Javascript, HTML, and CSS. Vue.js is a Javascript library for building the user interface. It is similar to React which can help developers to manage the data model and their HTML components. Typescript is a typed superset of Javascript maintained by Microsoft. We use Cytoscape.js[4], an open-source project with many useful layout plugins, as our graph library.

This guide focuses on how to extend the FPP Graphical Editor by integrating analyzers that analyze the topology of a FPP model. It also contains the definition of the CSS style file of the tool.

How to Run

1. Clone Github repository

a. Clone the FPP Graphical Editor repository from `https://github.com/the-jet-trio/fprime-editor/tree/master`

2. Install dependencies

a. Install Node.js[5].
b. Change to the source directory (`fprime-editor`) you just cloned.
c. Install the dependencies by running `npm install`.

[1] https://electronjs.org/
[2] https://vuejs.org/
[3] https://www.typescriptlang.org/
[4] http://js.cytoscape.org/
[5] https://nodejs.org/en/

3. Run the tool in development mode

a. Run `npm run dev` to run the tool with Chrome developer tool. This allows developers to debug more easily.

4. Build the tool

a. Run `npm run build` to build the tool.
b. The production package will be stored under `fprime-editor/build/`.

Adding a New Analyzer

1. Requirements of the analyzer

To integrate your model analyzer into the system, the analyzer should at least satisfy the following prerequisites:

- The analyzer shall be a standalone Java program that can be invoked via command line.
- For the output text, the analyzer shall print in to the standard output.
- For the graphical output, the analyzer shall produce the analysis result into a .css file that conforms to the style file definition, which can be read by the system. The style file definition will be discussed in the next section.

The following example shows how to run the Acme Rule Checker via command line. We have a folder containing:

- `Project/Simple.fpp`: A simple fpp file
- `Project/Acme-checker.jar`: The Acme Rule Checker

Under folder `Project`, run `java -jar ./acme-checker.jar . ./acme_output` to run the Acme Rule Checker. You will see something like

```
$ java -jar ./acme-checker.jar . ./acme_output
DEBUG: --------START OF OUTPUT--------
DEBUG: ===============================
ERROR: Simple.fpp line 56: conn1 in conn1
ERROR: Simple.fpp line 56: Target can only be input port
DEBUG: Generate acme_result.css at
       C:/fprime-editor/Project/acme_output
       time spend: 697ms
DEBUG: --------END OF OUTPUT--------
```

The analysis result, which is a .css file, will be stored at `Project/acme_output`. The output CSS file sets the selected connection to red:

```
// Project/acme_output/acme_result.css
#Simple_c1_pout-Simple_c2_pout { width: 4; line-color: red; }
```

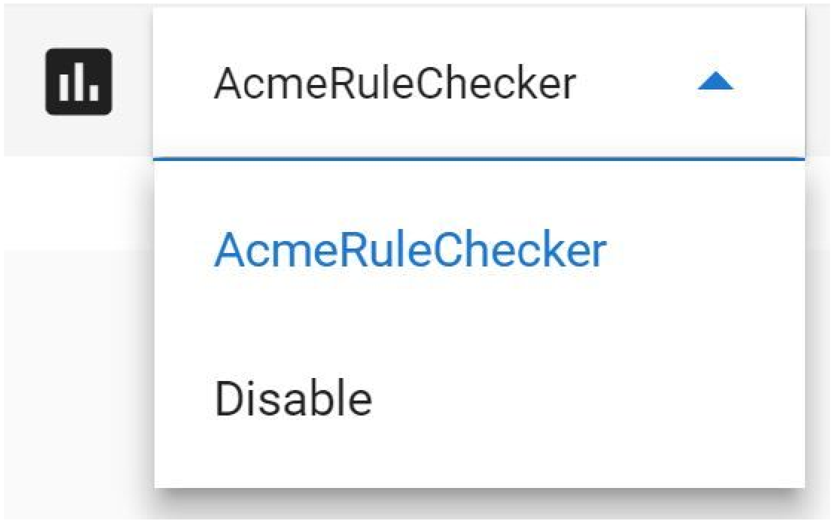
2. Integrating the Analyzer

1) Copy your analyzer to `fprime-editor/static`. (This is the default path; you can change this in `fprime-editor/static/config.json`.)
2) Open `fprime-editor/static/config.json`
3) `config.json` specifies the configuration settings of the analyzers. For example, the properties of the Acme Rule Checker looks like:

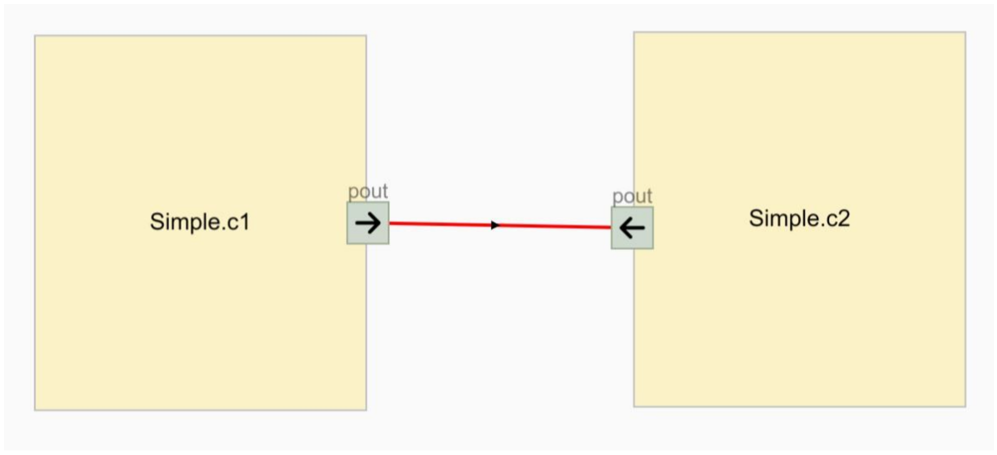
```
{
  "Analyzers": [
    {
      "Name": "AcmeRuleChecker",
      "Path": "${System}/acme-checker.jar",
      "Parameters": "${Project} ${Project}/acme_output",
      "OutputFilePath": "./acme_output/acme_result.css",
      "Type": "Rule Checker"
    }
  ],
  ...
}
```

   - `Name`: Name of the analyzer that will be shown on the UI
   - `Path`: Relative path of the analyzer. `${System}` refers to `fprime-editor/static` here.
   - `Parameters`: Input parameters for invoking the analyzer with command line. In this example, `${Project}` is the project path. The first parameter is the FPP project directory, and the second parameter is the output path of the analysis result.
   - `OutputFilePath`: The path where the analyzer stores the result.
   - `Type`: Type of the analyzer.

4) Add the properties of your analyzer to `config.json`, which will integrate the analyzer to the tool. For example, the following screenshot shows how the Acme Rule Checker is integrated.



5) Click ▮ to trigger the analyzer.
6) Click on the functional topology view to see the result. Take `Simple.fpp` as an example:



The connection between `Simple.c1` and `Simple.c2` is highlighted red because it is a violation of rules detected by the Acme Rule Checker.

Configuration File Definition

1. System-Level Configuration

The tool has two levels of configuration: system-level and project-level. The system-level configuration provides the default system configurations among all the projects. You can find the system configuration file at `fprime-editor/static/config.json`. It is a JSON file, and the fields are as follows:

- **Analyzers**: An array of the analyzer options.

    - **Analyzer option**: The configuration of a model analyzer.
        - ➢ **Name**: The name of the model analyzer which would be displayed on user interface.
        - ➢ **Path**: The path to the model analyzer executable.
        - ➢ **Parameters**: The command line parameters for running the model analyzer.
        - ➢ **OutputFilePath**: The path to the analysis result file which should be a valid style file (.css file). The system would read this file to load the analysis information.
        - ➢ **Type**: The type of the model analyzer. (Note: this property is not used currently.)
- **FPPCompilerPath**: The path to the FPP compiler. Note that the system contains a default FPP compiler. You do not need to change this property unless you want to use your own custom compiler.
- **FPPCompilerParameters**: The command line parameters for running the FPP compiler. You do not need to change this property unless you want to use your own custom compiler.
- **FPPCompilerOutputPath**: The path to the compiler result folder. Currently, the default FPP compiler will generate multiple representation files to the designated output folder. The system should use this path to load the compiled FPP model.
- **DefaultStyleFilePath**: The path to the default style file. See *Style File Definition* for more information.
- **ViewStyleFileFolder**: The path to the folder for storing the style file of each view. When saving the style of a view, the system should write the style file for that view in this folder.
- **AutoLayout**: An array of all the layout algorithms.
    - **Layout option**: The configuration of a layout algorithm.
        - ➢ **Name**: The name of the layout algorithm. The system will use this name to instantiate the layout algorithm in Cytoscape.js.
        - ➢ **Default**: Whether this layout algorithm is the default algorithm to use.
        - ➢ **Parameters**: The algorithm's parameters. Note: sometimes, the parameter of an algorithm may be a Javascript function. You can use the following syntax to create that function: "`(function(<args>) { <function body> })`".

2. Project-Level Configuration

The project-level configuration file is an optional JSON file defining the configurations of a project. The project-level configuration file shall have the name `config.json` and be placed under the root directory of the project, i.e., `<project_path>/config.json`. All the options in the project-level config file are optional, and the user-defined configurations will override the system-level configurations.

3. Path Resolving

All the paths in the configuration file will be resolved into the absolute path. There are two resolving rules:

- All the relative paths will be resolved from the project path. For instance, `"mystyle.css"` will be resolved as `"<project_path>/mystyle.css"`.
- Usage of `${System}` or `${Project}` placeholder. `${System}` will be replaced with the path to the `static` directory in the source repository (i.e., `fprime-editor/static`), and `${Project}` will be replaced with the project path.

Style File Definition

The system uses the standard CSS file to define the style information of the diagrams. There are two basic parts of a style definition: the selector and the properties.

1. FPrime Selectors

A selector is used to select the model elements in a diagram. You can change the fields of a selector to modify the appearance of the corresponding element(s).

`edge`	This selector selects all the edges in a diagram. For example:
	`edge {`
	`  line-color: rgb(0,0,0);`
	`  mid-target-arrow-shape: triangle;`
	`  mid-target-arrow-color: rgb(0,0,0);`
	`  width: 2px;`
	`}`

`.fprime-instance`	This selector selects all the component instances in a diagram. For example:
	`.fprime-instance {`
	`  shape: rectangle;`
	`  width: 400px;`
	`  height: 450px;`
	`  label: data(label);`
	`  font-size: 28px;`
	`  text-halign: data(label_hloc);`
	`  text-valign: data(label_vloc);`
	`  text-opacity: 1;`
	`  text-margin-y: 0px;`
	`  background-color: rgb(252,242,199);`
	`  border-width: 2px;`
	`  border-color: rgb(192,192,192);`

	`}`

`.fprime-port`	This selector selects all the ports in a diagram. For example:
	`.fprime-port {`
	`  shape: rectangle;`
	`  width: 50px;`
	`  height: 50px;`
	`  label: data(label);`
	`  font-size: 25px;`
	`  text-opacity: 0.5;`
	`  text-halign: data(label_hloc);`
	`  text-valign: data(label_vloc);`
	`  background-color: rgb(205,217,205);`
	`  background-image: data(img);`
	`  background-width: 70%;`
	`  background-height: 70%;`
	`  border-width: 2px;`
	`  border-color: rgb(158,173,145);`
	`}`

`.fprime-component`	This selector selects all the components in a diagram. For example:
	`.fprime-component {`
	`  shape: rectangle;`
	`  width: 400px;`
	`  height: 450px;`
	`  label: data(label);`
	`  font-size: 28px;`
	`  text-opacity: 0.5;`
	`  text-halign: data(label_hloc);`
	`  text-valign: data(label_vloc);`
	`  text-margin-y: 0px;`
	`  background-color: rgb(216,204,186);`
	`  border-width: 2px;`
	`  border-color: rgb(192,192,192);`
	`}`

`.fprime-component-<component type>`	This selector selects a specific component and all the instances of that component in a diagram. For example:
	`.fprime-component-Svc_ActiveLogger {`
	`  width: 500px;`
	`  height: 500px;`
	`}`

`.fprime-port-<port type>`	This selector selects all the ports of the given port type. For example:
	`.fprime-port-Fw_Log {`
	`  background-color: blue;`
	`}`

`.fprime-port-in`	This selector selects all the in-ports in a diagram. For example:
	`.fprime-port-in {`
	`  background-color: red;`
	`}`

`.fprime-port-out`	This selector selects all the out-ports in a diagram. For example:
	`.fprime-port-out {`
	`  background-color: green;`
	`}`

`.fprime-port-async`	This selector selects all the asynchronous ports in a diagram. For example:
	`.fprime-port-async {`
	`  background-color: red;`
	`}`

`.fprime-port-sync`	This selector selects all the synchronous ports in a diagram. For example:
	`.fprime-port-async {`
	`  background-color: blue;`
	`}`

`#<instance ID>`	This selector selects a single instance by its name in a diagram. For example, the following selector selects instance Ref.SG1:
	`#Ref_SG1 {`
	`  x: 217;`
	`  y: 103;`
	`}`

`#<instanceID>_<port name>-<instanceID>_<port name>`	This selector selects a single connection between two ports by its name. For example, `#Ref_ins1_p1-Ref-ins2_p2` selects the connection from port `p1` on instance `Ref.ins1` to port `p2` on instance `Ref.ins2`.
	`#Ref_ins1_p1-Ref-ins2_p2 {`
	`  width: 5;`
	`  line-color: green;`
	`}`

Note: For component names and port type names, the common format is `<namespace>.<type name>`. However, the '`.`' character has a special meaning in CSS. Thus, we replace '`.`' with '`_`'. For example, component `Comp1` in namespace `Ref` should be written as `Ref_Comp1` in the CSS files .

2. Style Properties

Since the system uses Cytoscape.js[6] as the graph rendering library, it supports all the style properties provided by Cytoscape. The connections between ports are represented as edges[7] in

[6] http://js.cytoscape.org/
[7] http://js.cytoscape.org/#style/edge-line

Cytoscape. The instances, ports, and components are represented as nodes[8] in Cytoscape. For instance, the following style definition changes all the in-ports to red:

```
.fprime-port-in {
    background-color: red;
}
```

Note that the annotations added latter will override the former ones. For example, the following style definition will set all in-ports to blue.

```
.fprime-port-in {
    background-color: red;
}
.fprime-port {
    background-color: blue;
}
```

3. Levels of Style Definition

The system uses three levels of style definitions: the system-level default style definitions, the project-level default style definitions, and the view-level style definition. The system-level default style definitions are effective among all the FPP projects. The project-level default style definitions are effective among all the views of a project. The view-level style definitions are effective for that particular view. The system loads the system-level default style first, then the project-level default style, and then the view style. According to CSS rules, the later definitions will override the previous ones.

[8] http://js.cytoscape.org/#style/node-body