Environment Framework

Finite State Machine (PoC)

User's Guide v1.0

Contents

1 The Concept

The State Machine Framework is a project on the proof-of-concept stage that was implemented by me and used in a lot of the projects. But at the same time, it is fully functional, although it has a number of limitations and shortcomings. This concept implements a mixture of Mealy and Moore's state machine concepts, which you will see later.

The package consists of two programs - an editor and a generator. The editor can create a state machine diagram in scxml format, but the extensions I made to it made this format incompatible with the original scxml, which you can find on the Internet. Also, the scxml format does not support information about the location of nodes, which is why it makes no sense to open the resulting diagrams in other editors - the states will not be located correctly.

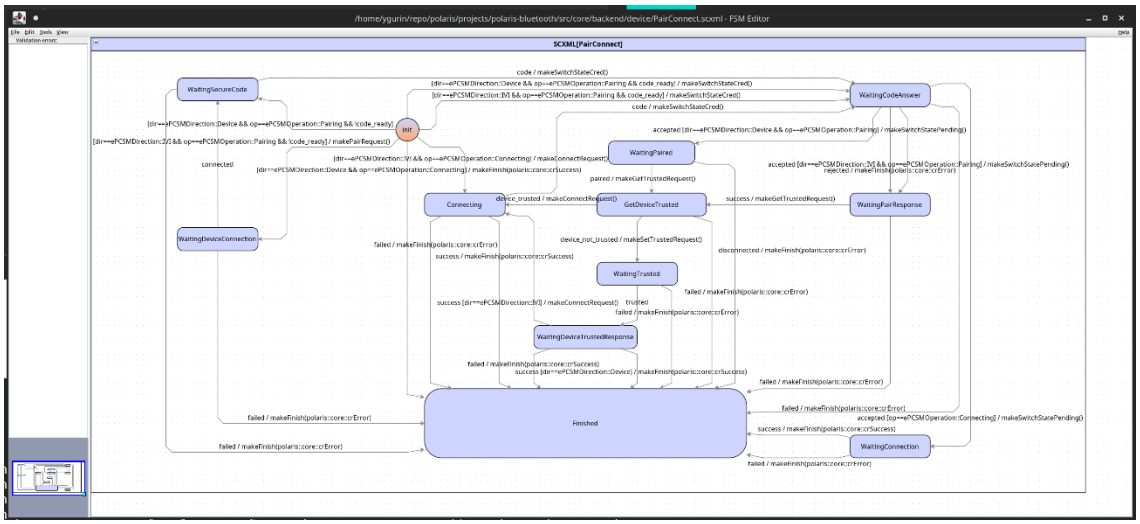

Figure 1

The generator is designed for generating the state machines from the models created in the editor.

2 Editor

2.1 Creating nodes

If you open the diagram and see that all the elements are clearly arranged chaotically and cannot be read, then pay attention to the '**Ignore stored layout**' checkbox in the '**File**' menu. This checkbox must be cleared. If it is set, then reset it and reopen the diagram.

There are four types of supported elements - *initial state*, *regular state*, *group state*, and *transition*. Every regular, group, or initial states supports two methods - **onEnter** and **onExit**. These methods will be generated by default. Transition supports an optional trigger, an optional condition, and optional set of transition handlers.

When developing your state machines, you must be attentive not only to the logic being implemented, but also to the appearance of the created diagram. Avoid unnecessary intersections, make the diagram as clear and logical as possible. Having a nicely constructed diagram is often just as important as having a well-functioning algorithm. This will make it easier to maintain and improve the extensibility of your system.

All types of nodes are created using the right mouse button and the context menu. To create your first state machine, start the editor and select '**New SCXML**' from the '**File**' menu. On an empty spot in the diagram, inside the diagram area, right-click and select '**Add node**'. The new state will appear. To convert it to another type of the node, for example, to an *initial state*, just right-click on it and select the required state from the list.

Do not use the standard element sizes suggested by the editor. They are not very suitable for making clear diagrams.

Create multiple states in the diagram. Remember that you can only have one initial state. If you use group states, then the diagram can have many initial states (the main one and one in each group).

Give names to your states. Remember, duplicate names are not allowed. To set the name, double-click on the node or select the '**Edit node**' item in the context menu:

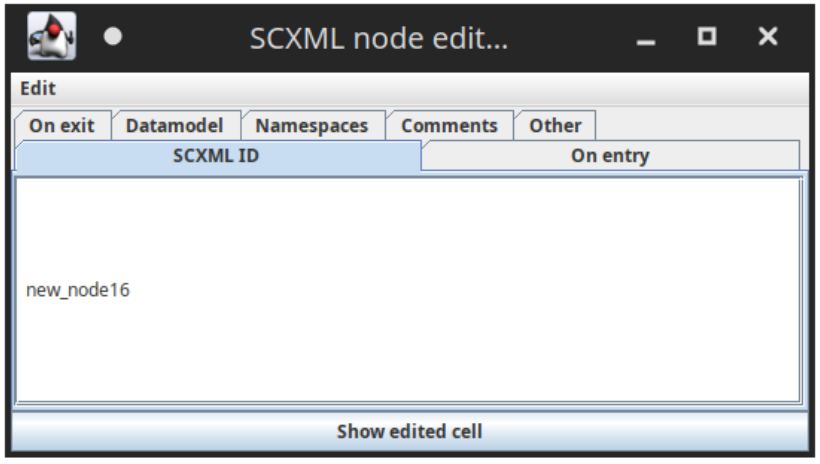

Figure 2

Don’t use other section except **SCXML ID** to give the name for the state.

2.2 Creating transitions

To create a transition, drag your mouse from the center of a state to another state. This will create a transition between them. An empty transition (autotransition) will be created by default. If you want to parameterize it, then right-click on it and select ‘**Edit Transition**’. You can also just double-click on it with the left mouse button.

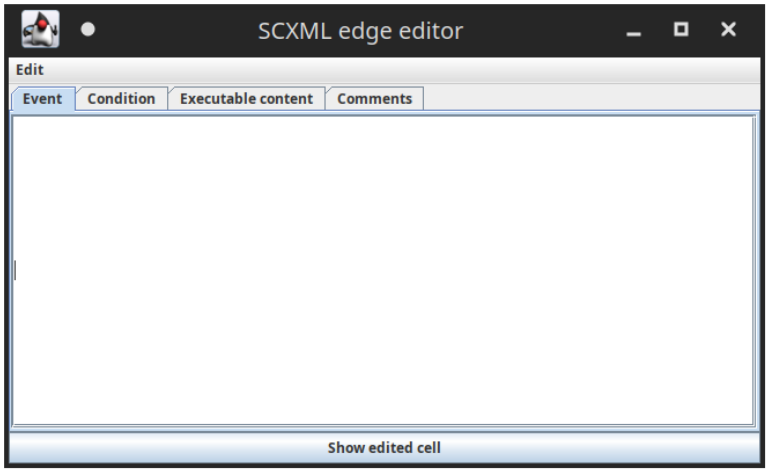

Figure 3

The editor has a limitation!
If you have created a transition between two nodes, and then renamed one of them, editing the transition will not be possible. To work around this limitation, either re-create the transition, or save and reopen the file with the diagram.

Each transition can be assigned with the Event (trigger), Condition and Executable content (transition handlers) fields.

2.2.1 Trigger

In the *Event* field, just fill in the trigger name if needed. Remember that the trigger name must not be the same as the name of other elements, including the data model fields. Also, this field can be empty. In this case you will obtain the autotransition.

2.2.2 Condition

In the *Condition* field, you can enter a condition according to the rules like those used in C++.

2.2.3 Transition handler

The *Executable content* field contains the names of transition handlers from those declared in the corresponding section. Enter these names as follows:

```
makeMyJob1()
makeMyJob2()
```

i.e. with brackets and each on a new line.

If you want to pass some parameter in the transition handler, you can specify it here, like:

```
makeMyJob1(5)
makeMyJob2(true)
```

In the case you want to pass the model value, use special syntax:

```
makeMyJob1(m.data)
```

2.2.4 Self-transitions

It is possible to create a self-transition. Just drag the transtition to the same state and the self-transition will be created. The next you must right click on the created dashed transition and select '**This cycle has target**' item. After the transition will be created and you can adjust it.

The editor has a limitation!
If you double-click on an empty transition and close the opened window, the editor will add an empty line to the Executable content field, which will cause the following element to appear on the diagram (Figure 4). Avoid such situations and remove this line. Once removed line will no longer be returned by the editor

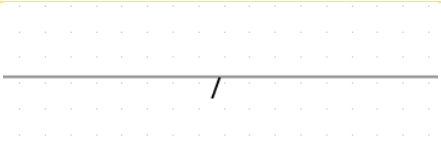

Figure 4

2.3 Editing state machine properties

To edit other chart properties, click on an empty area of the chart and select '**Edit node**'.

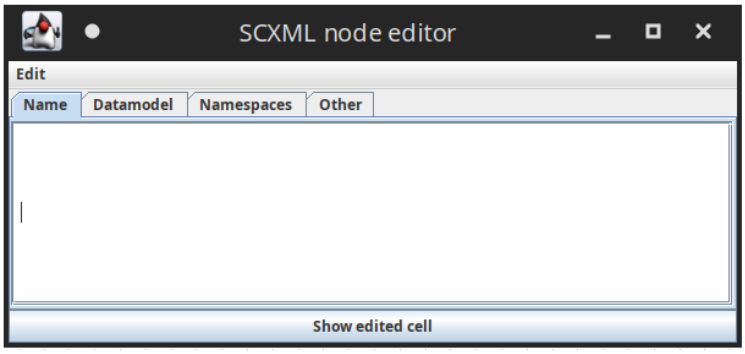

Figure 5

Don’t use **Namespace** section, it is not supported.

In the window that appears, you can edit several things described below.

2.3.1 The ‘Name’ tab

Specify an optional name for the chart.

2.3.2 The ‘Datamodel’ tab

In the **Datamodel** field specify the list of fields that will be available in the chart. The fields are set according to the C++ syntax:

```
int value;
std::string data;
```

The generator will generate the appropriate setters for all the model elements. The getters will not be generated! (it is due the ideology principles).

2.3.3 The ‘Other’ tab

In the **Other** field, a piece of xml is entered, in which you can specify two additional sections - handlers and header.

The editor has a limitation!
It is necessary to use escape symbols when you are editing elements using this program. The program is not escaping the symbols, so you must do it by yourself. Escape such symbols like “", ‘<’, '>’ and other if necessary. The standard XML escaping rules are working there, see the table below.

Special character	escaped form	get replaced by
Ampersand	&amp;	&
Less-than	&lt;	<
Greater-than	&gt;	>
Quotes	&quot;	“
Apostrophe	&apos;	'

2.3.3.1 Header section

Use this section to add your custom information to the generated header file. You can define your custom forwards, types and includes in this section.

For example, if you want to use the shared pointers in your state machine, you must specify:

```
<header>
#include &amp;lt;memory&amp;gt;
</header>
```

The <memory> header will be added to the generated file.

2.3.3.2 Handlers section

This section describes all the transition callbacks. You can specify the callback name and the parameters here. This is an example:

```
<handlers>
 <method name="makeSwitchStateCred"></method>
 <method name="makeSwitchStatePending"></method>
 <method name="makeGetTrustedRequest"></method>
 <method name="makeSetTrustedRequest"></method>
 <method name="makePairRequest"></method>
 <method name="makeConnectRequest"></method>
 <method name="makeFinish">
  <arg name="result" type="company::core::eCommandOperationResult"></arg>
 </method>
</handlers>
```

Here we have several transition handlers without parameters and one with `result` parameter and the type `company::core::eCommandOperationResult`. You can specify as many parameters as needed.

3 Generator

To generate the state machine just run the generator, it will print the help screen.

```
ygurin@ubuntu:~/fsm$ ./escxmlcc
Extended scxml FSM compiler, version "0.61"
Copyright (C) 2013 Jan Pedersen jp@jp-embedded.com
Copyright (C) 2014 Yuriy Gurin ygurin@outlook.com
Usage: ./escxmlcc [options] [input]
Options:
-h [ --help ] This help message
-i [ --input ] arg Input file
-o [ --output ] arg Output directory
-a [ --asyncs ] With Asyncs
-v [ --version ] Version information
```

The example how to generate the source code:

```
./escxmlcc -i mydiagram.scxml -o /path/to/the/products/dir
```

4 Annexes

https://github.com/aeore/escxmlcc

5 Change History

Version	Description	Status	Author	Date
1.0	First version	RELEASED	Yuriy Gurin (ygurin@outlook.com)	01 Nov 2023 г.