



L'ÉCOLE DE  
L'INNOVATION  
TECHNOLOGIQUE

## Esterel project report

**Iñigo AGUAS ARDAIZ**  
**Mido Mohamed MELLOULI**

Critical Application Development (IS-5103E)

2016-2017

International Master of Computer Science (IMC)

ESIEE Paris

# Table of contents

<b>Table of contents</b>	<b>2</b>
<b>Objectives and overview</b>	<b>3</b>
<b>Design &amp; Requirements</b>	<b>3</b>
General approach	3
Specifications	3
Regular specifications	3
Additional specifications	4
Platform environment	4
Esterel	4
Oracle VM VirtualBox	5
GitHub	5
GtkSourceView	5
<b>Implementation</b>	<b>5</b>
Esterel project structure	5
Tools.strl	5
Button.strl	6
Cabin.strl	6
Elevator.strl	7
C Project structure	7
<b>Validation interfaces</b>	<b>7</b>
Esterel Simulator testing and validation	7
C interface Testing and Validation	8
<b>Conclusions and future work</b>	<b>10</b>
<b>Appendix: user manual</b>	<b>11</b>
Compilation and execution	11
Simulation mode	11
C interface mode	12
<b>Appendix: Main panels of the implemented Esterel files in the Elevator project</b>	<b>13</b>

# 1. Objectives and overview

In this project, we have been invited to model and execute an elevator, in regards to its paper-based specification.

The report is organized as follows: Section 2 presents requirements and an Esterel based design for the elevator.

Section 3 details decomposition and algorithms used in the implementation of the Esterel project.

In Section 4, validation and tests of our outputs are presented.

Conclusions and future work are found in the last section, in Section 5.

Finally, a user manual (available at the Appendix A) of the project shall be included to describe how the project has been used. Also the main panels of the simulation mode in Appendix B.

## 2. Design & Requirements

### 2.1. General approach

An elevator is a vehicle which moves people between floors of a building. Usually it uses a software to be controlled and it is treated as an embedded system.

### 2.2. Specifications

#### 2.2.1. Regular specifications

For all the following, we consider the set  $i=\{1, 2, 3, 4\}$  which refers the floor number.

Our elevator is installed in a building of four floors, denoted as  $FLOOR_i$ . These floors are separated each by a distance of four meters, in this building. The lift has a velocity of 0.25 meter per second.

For each floor, there is a possibility to call the elevator. That can be done by pressing on a light button to call it, that we name  $CALL$ . This light button remains switched on when the elevator call has been requested, while it's still moving. As soon as the elevator has reached its final destination, the light in consideration will be switched off.

A panel at the top of the door of the elevator on the waiting floor indicates the  $CURRENT$   $FLOOR$ . That is applied for each of the floors of the building, as well as inside the cabin.

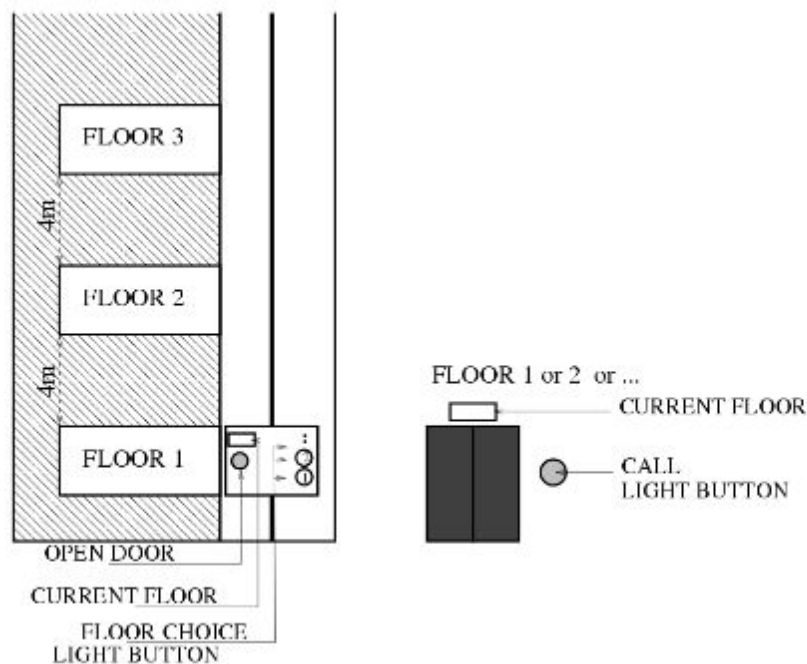
When the elevator is ready to go up or down to move to another floor, the doors of cabin are closed.

Speaking of the cabin, apart from the display mentioned in the previous paragraph, it has four light buttons to indicate our floor target when we are inside the cabin, which is denoted **BUTTON<sub>i</sub>**.

When we press on one or few of them, they remain lighted until each of their corresponding floors have been reached. The process of closing and opening the door of the cabin when reaching each floor, or when opening the door at its initial state, is done within two seconds.

Another available button inside the cabin is the **OPEN DOOR** which allows us to open the door of the cabin in case we are inside and would like to exit, and the door is closed.

The next figure illustrates the elevator system:



*Figure 1: illustration of the elevator with consisting information about its content*

### 2.2.2. Additional specifications

We have given some additional specifications that we think they are essential for the good functioning of our elevator.

In fact, the elevator moves initially from the first floor. Moreover, it starts with doors which are already closed. It also begins with all the buttons which are released. Thus, their lights are turned off, as no action has done on these.

## 2.3. Platform environment

### 2.3.1. Esterel

Esterel is a programming language and a compiler which translates a conceived program into finite-state machines. Its compilation makes a C files, which you can execute.

### 2.3.2. Oracle VM VirtualBox

It's an open-source solution for running other operating systems virtually. We install all the needed software inside with the intention the make a portable development.

### 2.3.3. GitHub

GitHub is a web-based repository hosting service. It offers all of the distributed version control and source code management (SCM) functionality of Git as well as adding its own features. We stored all the files of the project in a repository, with the intention of making development and sharing easier and also in order to be able to show different revisions.

### 2.3.4. GtkSourceView

GtkSourceView is a portable C library that extends the Debian multiline text editing feature with support for configurable syntax highlighting. This feature was used in order to introduce the Esterel grammar and colorize it.

## 3. Implementation

Before creating the virtual machine which utilizes the Esterel language, the project implementation has been realized in the beginning on a 64-bit HP Personal Computer, using the Debian Operating System. Also, it was developed in the virtual machine that has been setup to implement this project<sup>1</sup>.

Further implementation instructions can be found on the following link:

<https://github.com/iaguas/esterel-elevator>

In this section, we consider the implementation structure of two different development phases, starting with the Esterel structure, to the C coding one.

### 3.1. Esterel project structure

Four modules have been conceived for the realization of the modeling elevator: *Buttons*; *Cabin*; *Elevator*; and *Tools* modules.

---

<sup>1</sup> The VM can be donwloaed from:

<https://drive.google.com/open?id=0BzMEc-aXNUQed0MwU240WVd2TDQ>

### 3.1.1. Tools.strl

The module of Tools manages the some used sub modules of the elevator system, regarding some set of triggers / actions, and few created runnables.

This file has been created in order to avoid the coding redundancy of repeated lines of codings.

We have created in it four sub-modules which are executed according to the execution test case of the test scenario:

- OpenDoor: sub-module for opening the door of the cabin.
- CloseDoor: sub-module for closing the door of the cabin.
- FloorProcessingTime: sub-module to process the time to go from one floor to another.
- DoorAwaitingTime: sub-module for the await of one second spent.

### 3.1.2. Button.strl

The *Button* module manages the behavior of all existing buttons in the whole system, starting with the buttons of calling floors, which are existing on each one of them, and the ones inside the cabin to specify the floor target. Also, some other buttons are those of opening and closing the door of the cabin.

A series of actions are tested in order to find out the actual floor. For memory management, and in order to save all the targets that we would like to reach, we have established a way to identify whether the floor target in consideration is impacted by such movements between targeted floors.

We therefore assign to each floor number a state *stateFi* (where  $i \in \{1, 2, 3, 4\}$ : the floor number), whose result is false or true, to confirm whether the floor in consideration has been called or not. This will be considered later to move up or down, according to pressed targeted buttons.

Speaking of moving up or down the elevator, their corresponding set of instructions have been split into two sub-blocks.

If for instance, we already know that the floor number four has been called, and that the current position of the cabin is different from that calling position, we are pretty sure that the cabin would have to move up and go to the last floor.

Otherwise, the cabin would have to go down; in case another floor target has been introduced with the number four from the beginning, we will also say that the state of the floor number one is active and will say that the next floor to be processed is this floor.

This strategy is the same that we apply for each analysed floor.

### 3.1.3. Cabin.strl

The cabin module manages the movement behavior of the cabin according to its floor position; the opening and closing of its doors.

Based on one floor target *FLOORTARGET* (which will be applied to all of them as discussed in the previous section for memory management), it will be compared to the current *floor* where the cabin is situated. According to that, we make a decision, in closing doors (in case they were already open for some reasons), and go up or down.

In case we have reached the goal, we execute the set of actions related to the opening of the door. We apply the same strategy likewise to opt for the direction that the cabin is taking to go to the targeted floor.

In this module also, we satisfy in each sub-block the different requirements related to the duration that the process takes in order to move from one floor to another.

For each transitional floor, and in order to reach the final one, we always update output values and display the current floor, until we finally open the door when it's reached, and let the users go out from the cabin, and close the door. Starting from this position for further movings, we shall say that the elevator will be on stand-by.

### 3.1.4. Elevator.strl

This module is the main of the program for the running, with labeled namings, to run the *Cabin* and *Buttons* modules.

## 3.2. C Project structure

The Esterel compiler has been used in order to generate one part of our software implementation. C-codes have been generated, while other aspects have been coded manually; The C Project structure shall be utilized for the C interface, with some basic implementation.

Two files are considered for this part:

- *elevator\_main.c*: which contains all the functions for the interface with Esterel. It was developed by us.
- *elevator.c*: generated code with the makefile dependency. It contains the implementation for instance of outputs of the doors closed, or doors open, or the emission of the light related to the call button of the floor number *i* (where  $i \in \{1, 2, 3, 4\}$ ) and so on ...

## 4. Validation interfaces

We have two produced outputs from the implementation steps, that need to be evaluated; Esterel Simulator and the terminal interface.

### 4.1. Esterel Simulator testing and validation

We started in the beginning by testing and validating the esterel project at each time that we add new constraints related to the good functioning of the elevator.

As mentioned in the next panel of the Esterel simulator, based on our inputs and outputs, we set our inputs and establish some test cases. We also see the valued output of the current floor, as well as the pure outputs and make sure for instance whether the door has been opened, or closed, or if one of the light buttons is switched on or off.

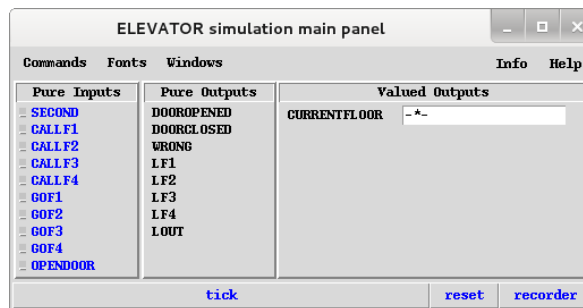


Figure 2: Elevator simulation main panel for inputs and outputs

For the following panel, we can follow the track of our simulation when we are reproducing a test case, which will be highlighted in red. That means that the current operation is done in that specific command line. For each esterel file, we have its main panel. We will only be interested for the following, on the Elevator Main Panel related to its esterel file. As for the other Esterel files, they can be used for Debugging mode. They are available at Appendix B.

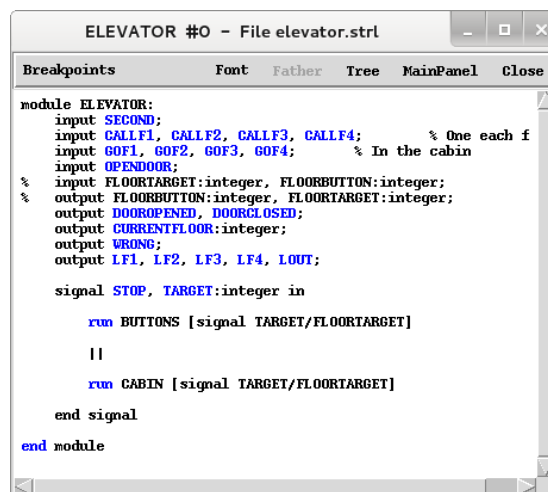


Figure 3: Elevator main panel file simulator

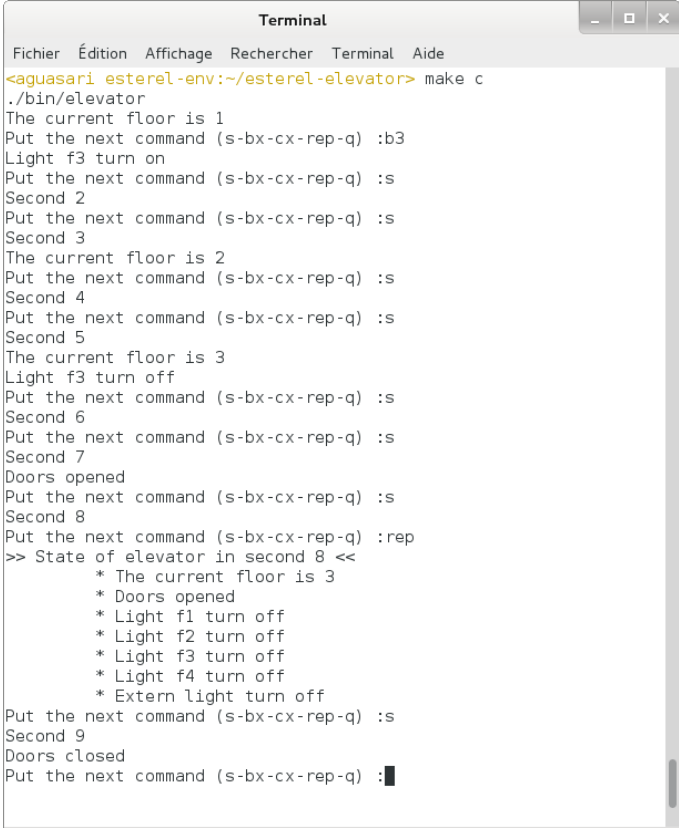


## 4.2. C interface Testing and Validation

Here in this paragraph, as shown as well in the following figure, we have an terminal interaction for monitoring the elevator, using the C interface.

Based on the appendix A related to the user manual of the project, it's good to remember that the initial state of the cabin is the first floor, as previously stated in additional requirements.

We have reproduced a test case as it's shown in the following figure;



```
Terminal
Fichier  Édition  Affichage  Rechercher  Terminal  Aide
<aguasari esterel-env:~/esterel-elevator> make c
./bin/elevator
The current floor is 1
Put the next command (s-bx-cx-rep-q) :b3
Light f3 turn on
Put the next command (s-bx-cx-rep-q) :s
Second 2
Put the next command (s-bx-cx-rep-q) :s
Second 3
The current floor is 2
Put the next command (s-bx-cx-rep-q) :s
Second 4
Put the next command (s-bx-cx-rep-q) :s
Second 5
The current floor is 3
Light f3 turn off
Put the next command (s-bx-cx-rep-q) :s
Second 6
Put the next command (s-bx-cx-rep-q) :s
Second 7
Doors opened
Put the next command (s-bx-cx-rep-q) :s
Second 8
Put the next command (s-bx-cx-rep-q) :rep
>> State of elevator in second 8 <<
    * The current floor is 3
    * Doors opened
    * Light f1 turn off
    * Light f2 turn off
    * Light f3 turn off
    * Light f4 turn off
    * Extern light turn off
Put the next command (s-bx-cx-rep-q) :s
Second 9
Doors closed
Put the next command (s-bx-cx-rep-q) :█
```

Figure 4: C interface of the C Project of the elevator

We will be interested in the first command of the interface that we have executed (using the `bx` command) and the last but one command using the `rep` command.

We would like to go to the third floor (via the command `b3`) → As an output, the button of the target floor (third one) has been turned on. As one instruction has been done, the value of `SECOND` will be incremented.

The second command to be analysed as mentioned is the `rep` command in our list of adopted commands; this will generate a report of the state of the elevator → it will give us useful information about the different lighting buttons of the different floors, and the state of the doors of the cabin, as well as the current floor where the cabin is situated.

## 5. Conclusions and future work

The used software environments have allowed us to implement and monitor an elevator, using Esterel commands. The established platform was a collection of different frameworks and tools, to satisfy the basic functional requirements available on the project statement.

Other requirements had to be carefully added and taken into consideration, for a better performance of the elevator, which are presented in this report, to fulfill all the needs and good features of the elevator. All of these have been simulated and then tested into a terminal command.

Esterel was a "valve" to implement the know-how of such an equipment.

Workflow from the introduction of the course to switch to different practical sessions, and to finish up with being devoted to work on a project was quite interesting as you gain knowledge and become quite confident after all enormous efforts which led to fruitful results.

One part hard was to work on our own machines. For that reason, we conclude that the easiest way to avoid these problems is to prepare a virtual machine. As the same idea, to improve the readability, we develop the grammar for the GtkSourceView.

Also, because of the changing of paradigm that Esterel makes, some bugs was so hard to correct them. To be exactly, all of those who have to been about the infinites loops was the hardest to solve.

Further work based on our Esterel work can be done in order implement and create from scratch a Graphical User Interface of the command terminal we have been done so far, for a nicer interaction with the user.

# A. Appendix: user manual

## I. Compilation and execution

There is a makefile programed to make all in a computer with Esterel installed. There are several options of makefile:

- Simulation mode:
  - simul-comp: Compilation for using the simulation mode.
  - simul32: Simulation mode for 32 bits core.
  - simul64: Simulation mode for 64 bits core.
- C interface mode:
  - c-comp: Compilation of all finales for using the C interface mode.
  - c: Option to remove binary files and other generated files.
- Cleaning mode:
  - clean: Option to remove binary files.
  - cleanall: Option to remove binary files and other generated files.

## II. Simulation mode

For all what is below, we consider that  $i$  is a set where  $x=\{1, 2, 3, 4\}$ , standing for the number of the floor. Also, the `SECOND` is a naming for the base unit of time, that we have been calling for each passing second.

After compilation and execution (with the correct core instructions), you will discover a window with xes simulator. In it you will discover all the signals of the program that can be used to control or minotor the elevator:

- *SECOND*: Every second that should pass, the user can send a signal to specify that.
- *CALLFx*: All the buttons that it can be discovered on the external part of the elevator, at each floor). As an example, *CALLF1* represents the call of the first floor.
- *GOFx*: All the buttons that it can be discovered in the cabin for each floor..
- *OPENDOOR*: button for door opening in that moment (if possible, depending on the different constraints of the lift).

In order to find out about the state of elevator, some output signals could be received. In the next highlighted dots, we mention some of their explanations:

- *DOOROPENED*: This signal is received each time that the door is starting to open.
- *DOORCLOSED*: This signal is received each time that the door is starting to close.
- *CURRENTFLOOR*: This is a valued signal that is received each time that you reach a new floor. It's displayed not only in the cabin but even on each floor, to give an indication to the user who is waiting what is the actual position of the cabin. It is received just when you arrived, even before open doors.
- *LFx*: The light of button of  $i$ th floor has changed the state. As soon as the result is reached, with a light button turned on, it will turn off.

- *LOUT*: This has the same idea as *LFi*, but it's actually application to all buttons on the each waiting floor, for calling the lift, from where we are.
- *WRONG*: This is a debug signal, to know if we are taking a bad way in moving. If it appears, it means that the elevator might present a software bug, and an initialization/reset is needed to make it operational once again.

The first thing that you should do is to pass a tick to initialize all the elevator. Then, you are free to try to control as you wish.

### III. C interface mode

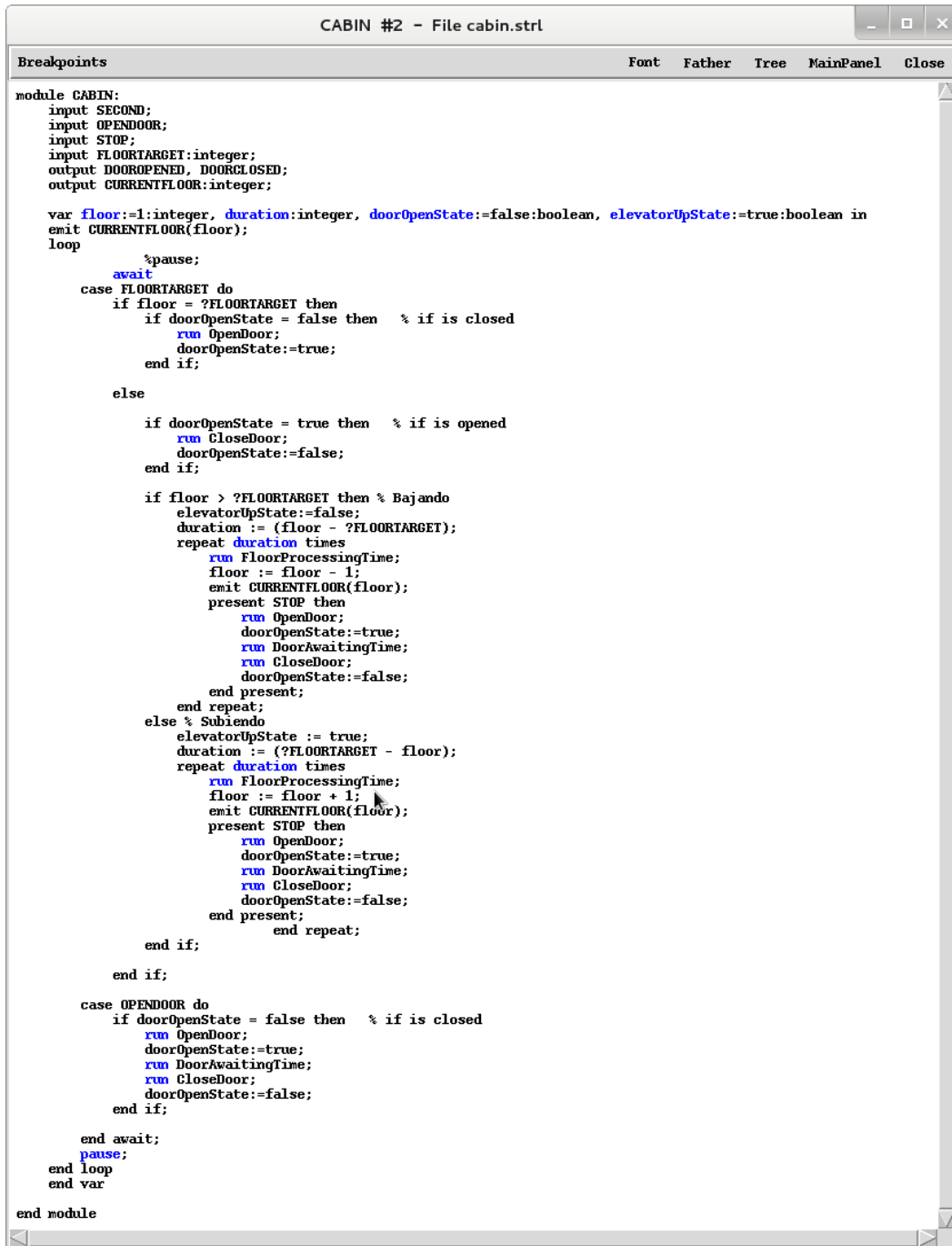
After compiling and executing, you should use the instruction *make c*.

When you do this, the C interface will show up. In it you should use some commands to manage the interface. As concretely, you have these ones:

- *s*: to send a *SECOND* signal
- *cx*: to send a *CALLFx* signal.
- *bx*: to send a *GOFx* signal.
- *od*: to send an *OPENDOOR* signal.
- *rep*: to take a report of the state of the elevator in that moment.
- *q*: to finish the program.

All the output signals could be seen at the moment that are received, and also with the report that we send to retrieve information about the current state.

## B. Appendix: Main panels of the implemented Esterel files in the Elevator project



```
module CABIN:
  input SECOND;
  input OPENDOOR;
  input STOP;
  input FLOORTARGET:integer;
  output DOOROPENED, DOORCLOSED;
  output CURRENTFLOOR:integer;

  var floor:=1:integer, duration:integer, doorOpenState:=false:boolean, elevatorUpState:=true:boolean in
  emit CURRENTFLOOR(floor);
  loop
    %pause;
    await
    case FLOORTARGET do
      if floor = ?FLOORTARGET then
        if doorOpenState = false then % if is closed
          run OpenDoor;
          doorOpenState:=true;
        end if;
      else
        if doorOpenState = true then % if is opened
          run CloseDoor;
          doorOpenState:=false;
        end if;

        if floor > ?FLOORTARGET then % Bajando
          elevatorUpState:=false;
          duration := (floor - ?FLOORTARGET);
          repeat duration times
            run FloorProcessingTime;
            floor := floor - 1;
            emit CURRENTFLOOR(floor);
            present STOP then
              run OpenDoor;
              doorOpenState:=true;
              run DoorAwaitingTime;
              run CloseDoor;
              doorOpenState:=false;
            end present;
          end repeat;
        else % Subiendo
          elevatorUpState := true;
          duration := (?FLOORTARGET - floor);
          repeat duration times
            run FloorProcessingTime;
            floor := floor + 1;
            emit CURRENTFLOOR(floor);
            present STOP then
              run OpenDoor;
              doorOpenState:=true;
              run DoorAwaitingTime;
              run CloseDoor;
              doorOpenState:=false;
            end present;
          end repeat;
        end if;
      end if;
    case OPENDOOR do
      if doorOpenState = false then % if is closed
        run OpenDoor;
        doorOpenState:=true;
        run DoorAwaitingTime;
        run CloseDoor;
        doorOpenState:=false;
      end if;
    end await;
    pause;
  end loop
end var
end module
```

Figure 5: Main panel related to the Esterel Cabin file

The screenshot shows a window titled "BUTTONS #1 - File buttons.strl". The window has a menu bar with "Breakpoints", "Font", "Father", "Tree", "MainPanel", and "Close". The main panel displays the following code:

```

module BUTTONS:
  input CALLF1, CALLF2, CALLF3, CALLF4;      % One for each floor
  input GOF1, GOF2, GOF3, GOF4;             % In the cabin
  input OPENDOOR;      % In the cabin
  input SECOND;
  input CURRENTFLOOR:integer;
  output FLOORTARGET:integer;
  output WRONG, STOP;
  output LF1, LF2, LF3, LF4, LOUT;
  output DOOROPENED, DOORCLOSED;

  var stateF1:=false:boolean, stateF2:=false:boolean, stateF3:=false:boolean, stateF4:=false:boolean, elevatorFree:=true:bool
  nextFloor:=1:integer in
    loop
      % One for each floor
      present CALLF1 or GOF1 then
        stateF1 := true;
        present GOF1 then
          emit LF1;
        end present;
      end present;
      present CALLF2 or GOF2 then
        stateF2 := true;
        present GOF2 then
          emit LF2;
        end present;
      end present;
      present CALLF3 or GOF3 then
        stateF3 := true;
        present GOF3 then
          emit LF3;
        end present;
      end present;
      present CALLF4 or GOF4 then
        stateF4 := true;
        present GOF4 then
          emit LF4;
        end present;
      end present;

      present CURRENTFLOOR then
        if ?CURRENTFLOOR = 1 then
          if stateF1 then
            stateF1 := false;
            emit LF1;
            emit STOP;
          end if;
        elsif ?CURRENTFLOOR = 2 then
          if stateF2 then
            stateF2 := false;
            emit LF2;
            emit STOP;
          end if;
        elsif ?CURRENTFLOOR = 3 then
          if stateF3 then
            stateF3 := false;
            emit LF3;
            emit STOP;
          end if;
        elsif ?CURRENTFLOOR = 4 then
          if stateF4 then
            stateF4 := false;
            emit LF4;
            emit STOP;
          end if;
        else
          emit WRONG;
        end if;
        if ?CURRENTFLOOR = nextFloor then
          elevatorFree := true;
        end if;
      end present;

      if elevatorFree and (stateF1 or stateF2 or stateF3 or stateF4) then
        % SOMETHING TO DO WITH BUTTONS
        if elevatorUpState then
          if stateF4 and nextFloor < 4 then
            emit FLOORTARGET(4);
            nextFloor:=4;
          end if;
        end if;
      end if;
    end loop;
  end

```

Figure 6: Main panel related to the Esterel Buttons file