Institut Supérieur de l'Aéronautique et de l'Espace
SUPAERO

SCADE – BE1
Hands-on Exercises Report

Implementation, Validation and Verification of a Temperature and Pressure Control System using SCADE

Student: Wendi Ding
Date: 05/12/2021

1 Goal of the Project

In this project, we manipulate the concepts of the Lustre synchronous language via the SCADE tool. 3 steps are followed:

- Design: switching from a textual high-level specification to a detailed specification, i.e. SCADE formalism data flow and state machines.
- Validation using simulation tools.
- Verification of the validation using model test coverage

2 Specifications

A controller is connected to two sensor: pressure and temperature. An alarm is triggered each time the temperature or pressure threshold is exceeded. The alarm lasts 30 time units. If the temperature threshold is exceeded, the controller shall trigger a cooling via a fan. The fan remains on for 120 time units and then stops. The controller has temperature and pressure setpoints that are not to be exceeded, and the latter can be modified.

3 Analysis of the System

The analysis of the system follows a Model-Based Systems Engineering (MBSE) sequence and takes the following steps:

- Extract the functional and non-functional registers from the specifications.
- Define the functions of the system: create a use case. Associate the requirements with each use case.
- Define the sequence diagram for:
    - exceeding temperature
    - exceeding pressure

3.1 Assumptions and Requirement Diagram

The assumptions of the modelling consist of three aspects:

- Not Modelled:
    - The system has been correctly installed and initialized.
    - The shutdown procedure will not be modelled.
    - No breakdown of components is addressed in the model.
    - Maintenance is not addressed by the model.
- System:
    - The controller will never fail.
    - The controller will never run out of power.
- Environment:
    - Neither the temperature and pressure sensors nor the alarm nor the fan will ever fail.
    - The temperature and pressure sensors, the alarm and the fan will permanently be connected to the controller.
    - The temperature and pressure sensors the alarm and the fan will never run out of power.

The breakdown of requirement and the allocation of functions to components are shown in the requirement diagram in Fig.1 and the requirement matrix in Table 1. The context diagram and the functional architecture are demonstrated in Fig.2-3.

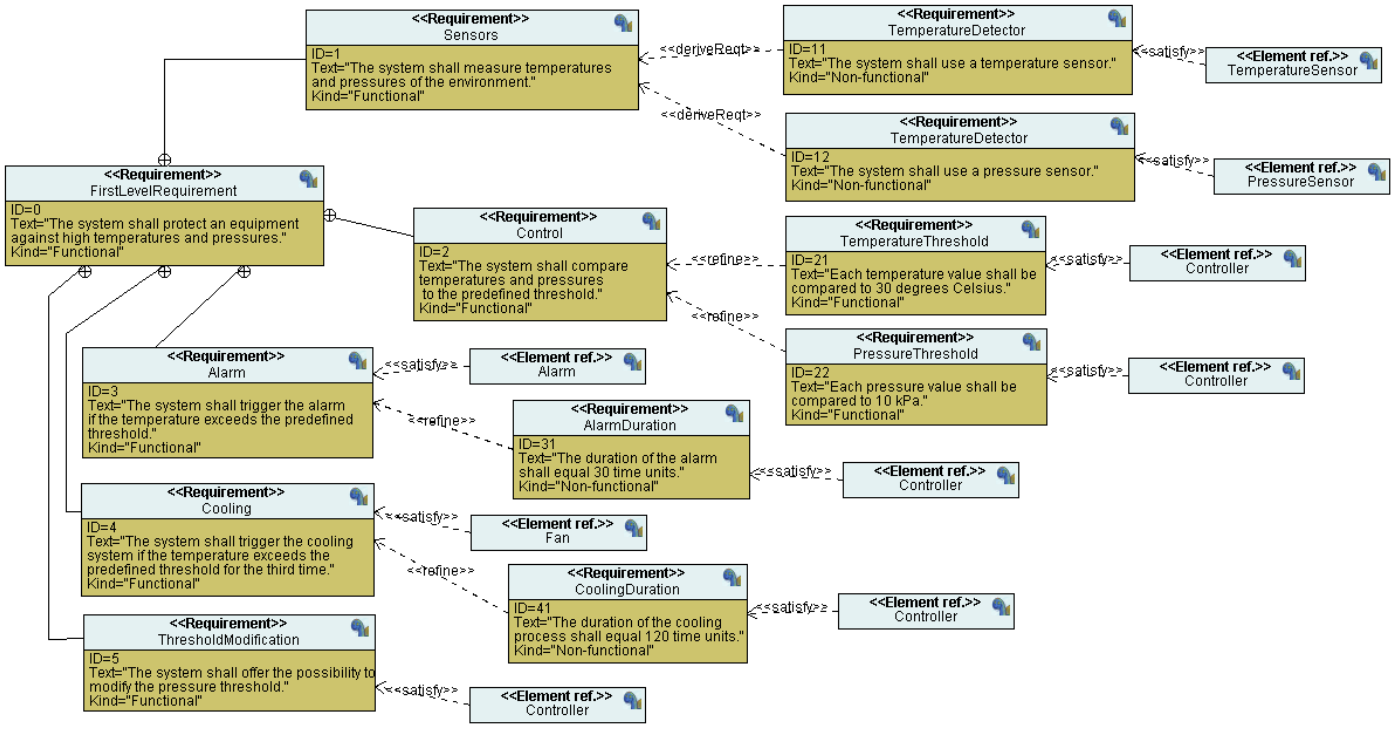


Figure 1: Requirement diagram

Table 1: Requirement Matrix

Req ID	Title	Link	Functional Unit
0	The system shall protect an equipment against high temperatures and pressures.	-	Controller
1	The system shall measure temperature and pressures of the environment.	1.1, 1.2	Sensors
1.1	The system shall use a temperature sensor.	-	Temperature Sensor
1.2	The system shall use a pressure sensor.	-	Pressure Sensor
2	The system shall compare temperatures and pressures to the predefined threshold.	2.1, 2.2	Controller
2.1	Each temperature value shall becompared to 30 degrees Celsius.	-	Controller
2.2	Each pressure value shall becompared to 10 kPa.	-	Controller
3	The system shall trigger the alarm if the temperature exceeds the predefined threshold.	3.1	Controller, Alarm
3.1	The duration of the alarm shall equal 30 time units.	-	Controller, Alarm
4	The system shall trigger the cooling system if the temperature exceeds the predefined threshold for the third time.	4.1	Controller, Fan
4.1	The duration of the cooling process shall equal 120 time units.	-	Controller, Fan
5	The system shall offer the possibility to modify the pressure threshold.	-	Controller

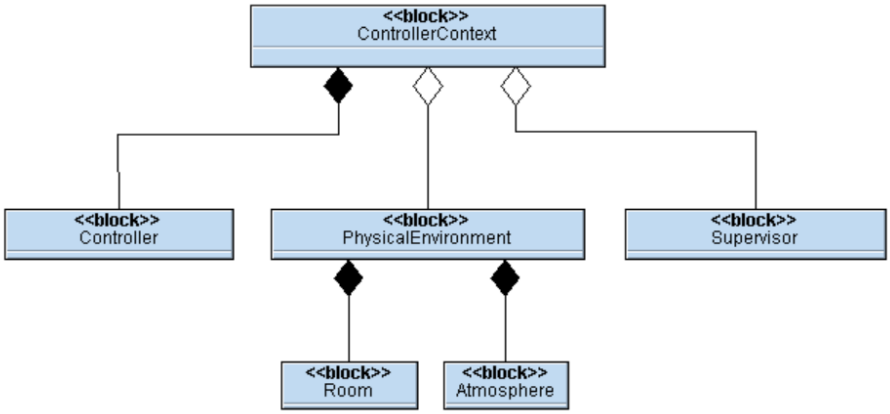


Figure 2: Context diagram

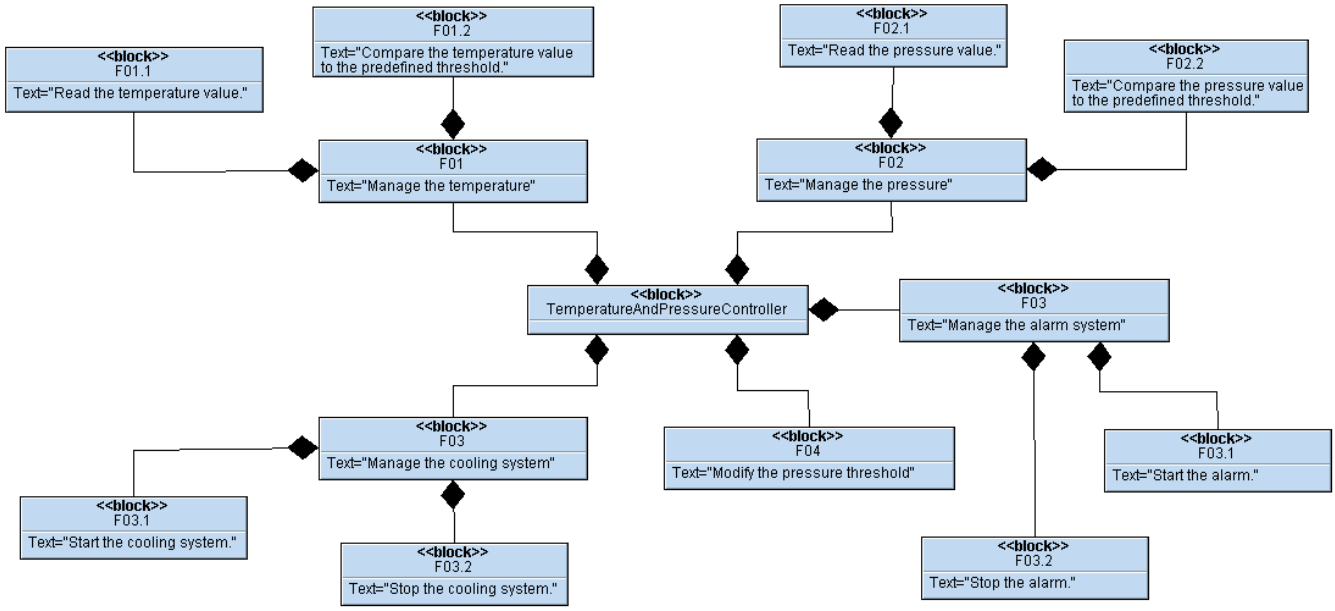


Figure 3: Functional architecture

3.2 Use Case Diagram

The use case diagram of the system shown in Fig.4. We associate the requirements with each use case in Fig.1 with the satisfy arrows.

The block diagram of the system is shown in Fig.5. The main components that share interfaces with the controller are the temperature sensor, the pressure sensor, the alarm and the fan. The sensors are in charge of temperature and pressure measurement respectively, and they send the temperature and pressure values to the controller upon request from the controller. Thus the temperature sensor offers the interface getTemperature() and the pressure sensor offers the interface getPressure() to the controller. The controller offers the interfaces temperature(float temperature) and pressure(float pressure) to receive the temperature and pressure values from the sensors. The start and stop of the alarm and the fan is also controlled by the controller. If the temperature or pressure threshold is exceeded, the controller will send startAlarm() signal to the alarm which leads the alarm to ringing. After 30 time units, the controller will send the stopAlarm() signal to the alarm to stop it. Similarly,

if the temperature threshold is exceeded, the controller will also send startFan() signal to the fan which leads the fan to working. After 120 time units, the controller will send the stopFan() signal to stop the fan. Thus the alarm offers the interfaces startAlarm() and stopAlarm() to the controller while the fan offers the interfaces startFan() and stopFan() to the controller. The interfaces of the system is shown in the class diagram and internal block diagram in Fig.6-7.

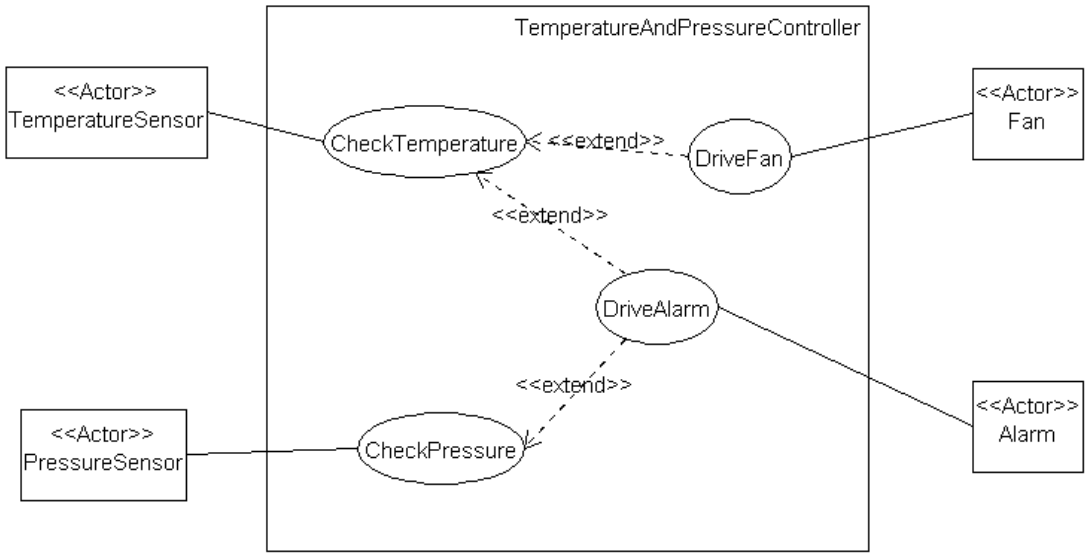


Figure 4: Use case diagram

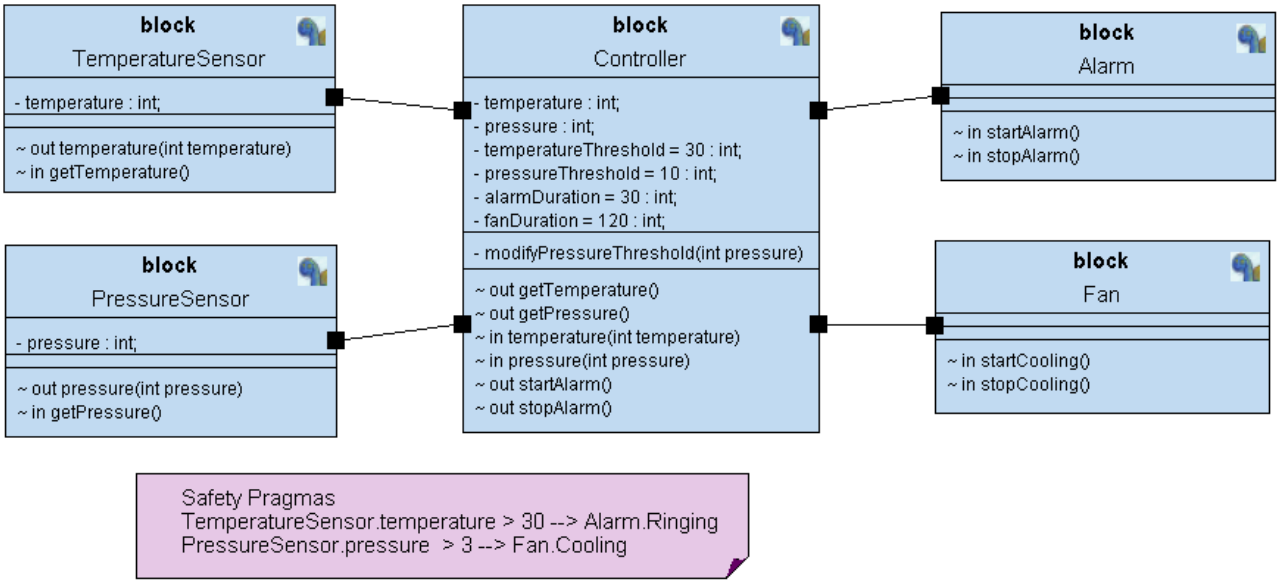


Figure 5: Block diagram

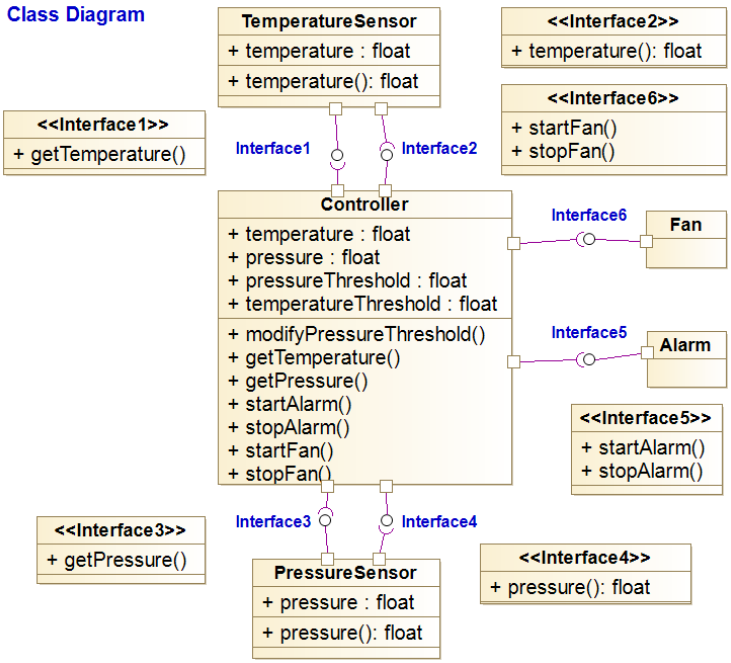


Figure 6: Class diagram and interfaces

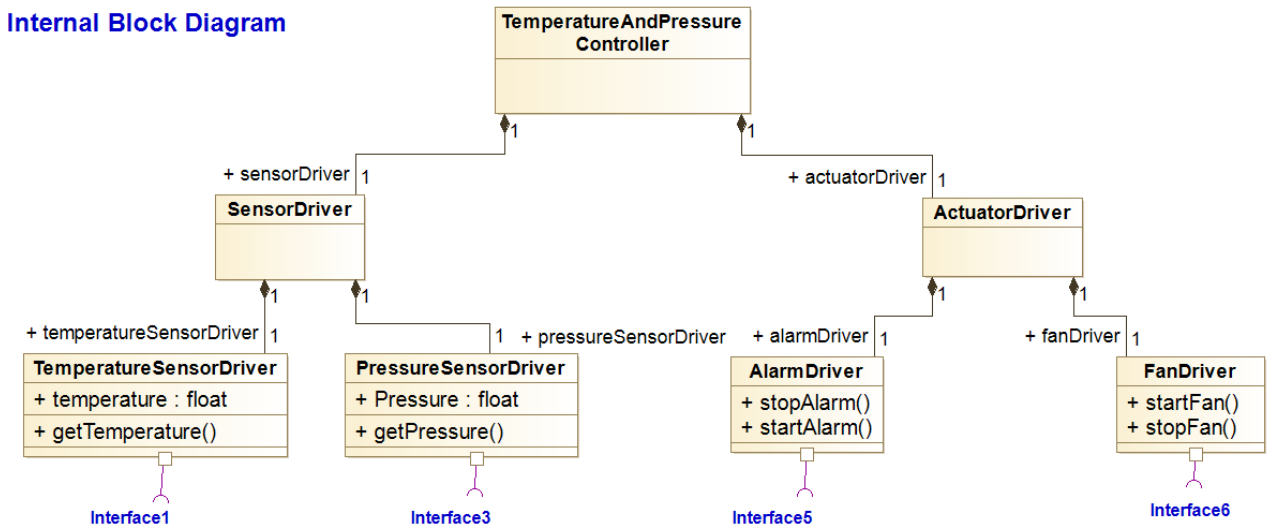


Figure 7: Internal block diagram of the temperature and pressure control system

3.3 Sequence Diagram

The sequence diagram is defined for three scenarios:

- OPT1: exceeding pressure (Fig.8)
- OPT2: not exceeding pressure nor temperature (Fig.8)
- OPT3: exceeding temperature (Fig.9)

In Fig.8, we can see that all actuators are initialized to the initial state at the beginning. Then the controller request temperature from the temperature sensor and request pressure from the pressure

sensor. The sensors send the controller the values respectively, say the temperature is 28 degrees and the pressure is 11 kPa (OPT1). Upon receiving the sensor values, the controller compares the values with the thresholds. The temperature and pressure thresholds are predefined as 30 degrees and 10 kPa respectively. Therefore, the pressure exceeds the pressure threshold and thus the controller sends startAlarm() to the alarm. The alarm is triggered during 30 time units before the controller sends stopAlarm() to stop it.

In OPT2 (Fig.8), the temperature is 29 degrees and the pressure is 9 kPa, which is below their thresholds. Therefore, nothing is needed to be done.

In OPT3 (Fig.9), the pressure threshold is modified from 10 to 12kPa. The temperature is 31 degrees and the pressure is 12 kPa. Therefore, the temperature exceeds the temperature threshold and thus the controller sends startAlarm() to the alarm. At the same time, the controller also sends startFan() to the fan. The alarm is triggered during 30 time units before the controller sends stopAlarm() to stop it. The fan is working during 120 time units before the controller sends stopFan() to stop it.

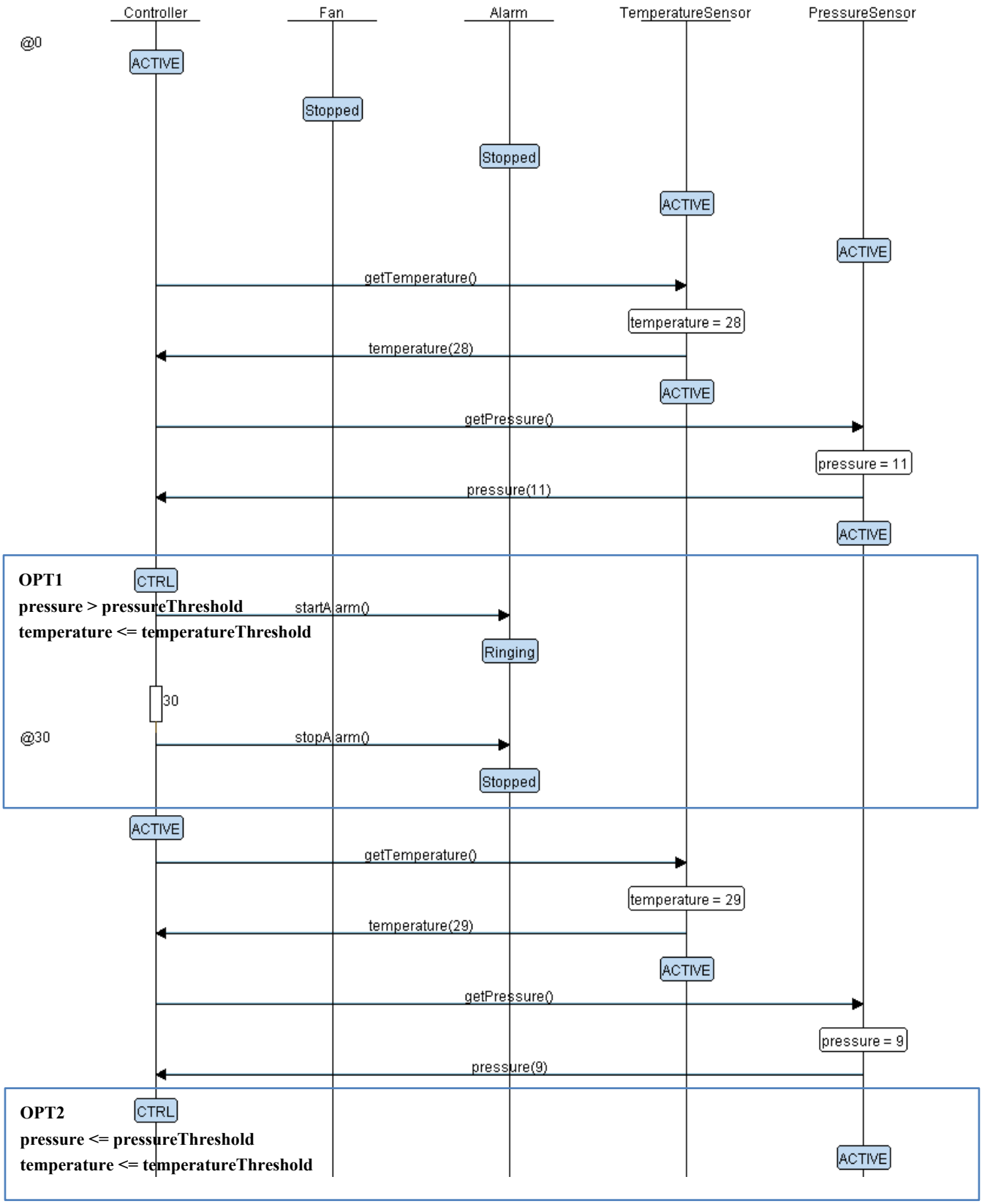


Figure 8: Sequence diagram (1)

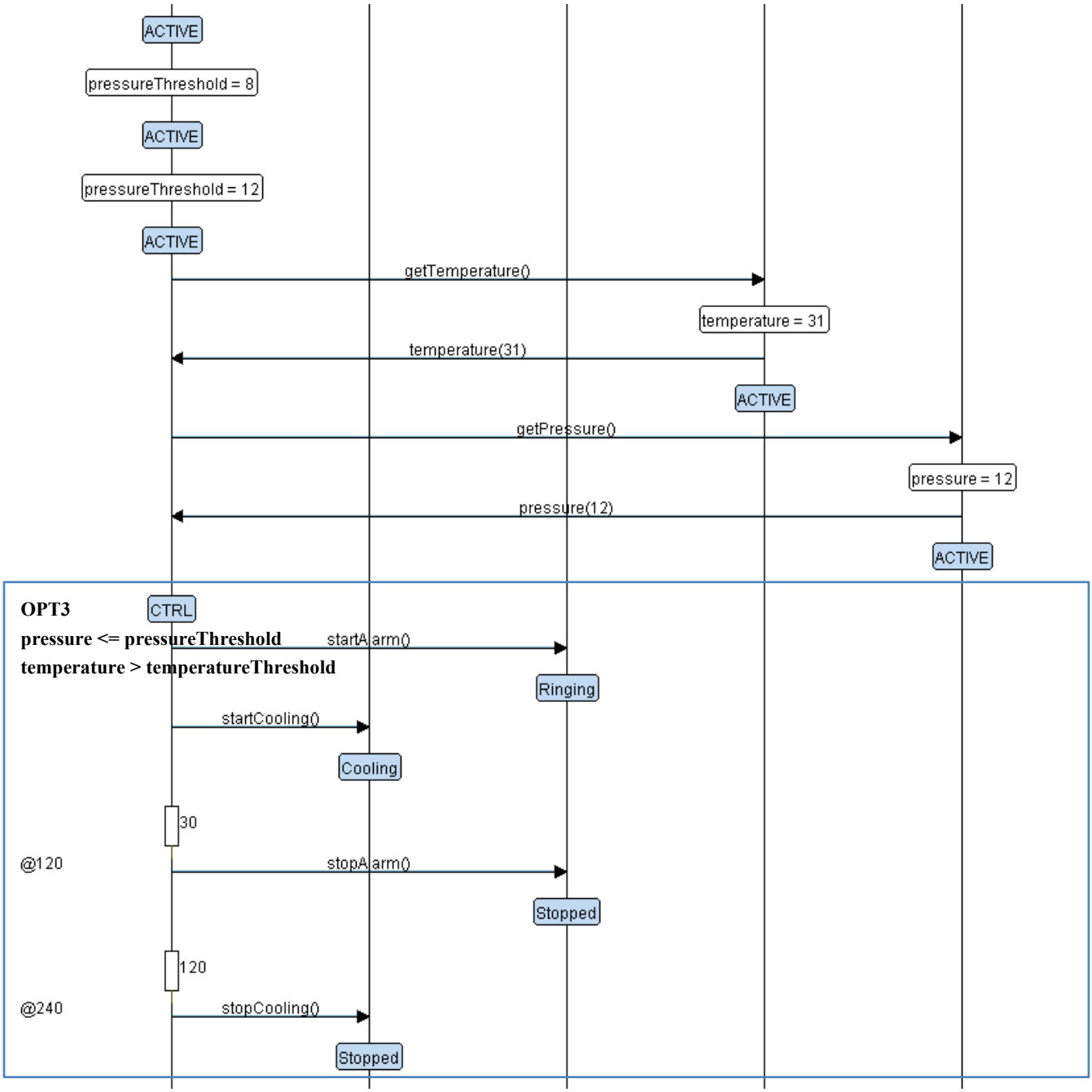


Figure 9: Sequence diagram (2)

4 Implementation in SCADE

The controller is implemented in Scade in Fig.10. The Controller operator takes as input 2 real streams, i.e. temperature and pressure, and returns 2 Boolean output streams, i.e. AlarmOnOff and FanOnOff. It containes two state machines, i.e. alarm controller and fan controller, taking responsibility of alarm and fan control respectively and the states depending on the local variables t_alart and t_alert. Two nodes are implemented twice respectively in the controller, i.e. CheckThreshold and SwitchOnOff. The CheckThreshold node is a comparator of the temperature or pressure values and the thresholds which generates t_Alert or p_Alert local signals. The

SwitchOnOff node contains a state machine which takes local variables startAlarm/Fan and stopAlarm/Fan as input and generates alarm/fanOnOff as output.

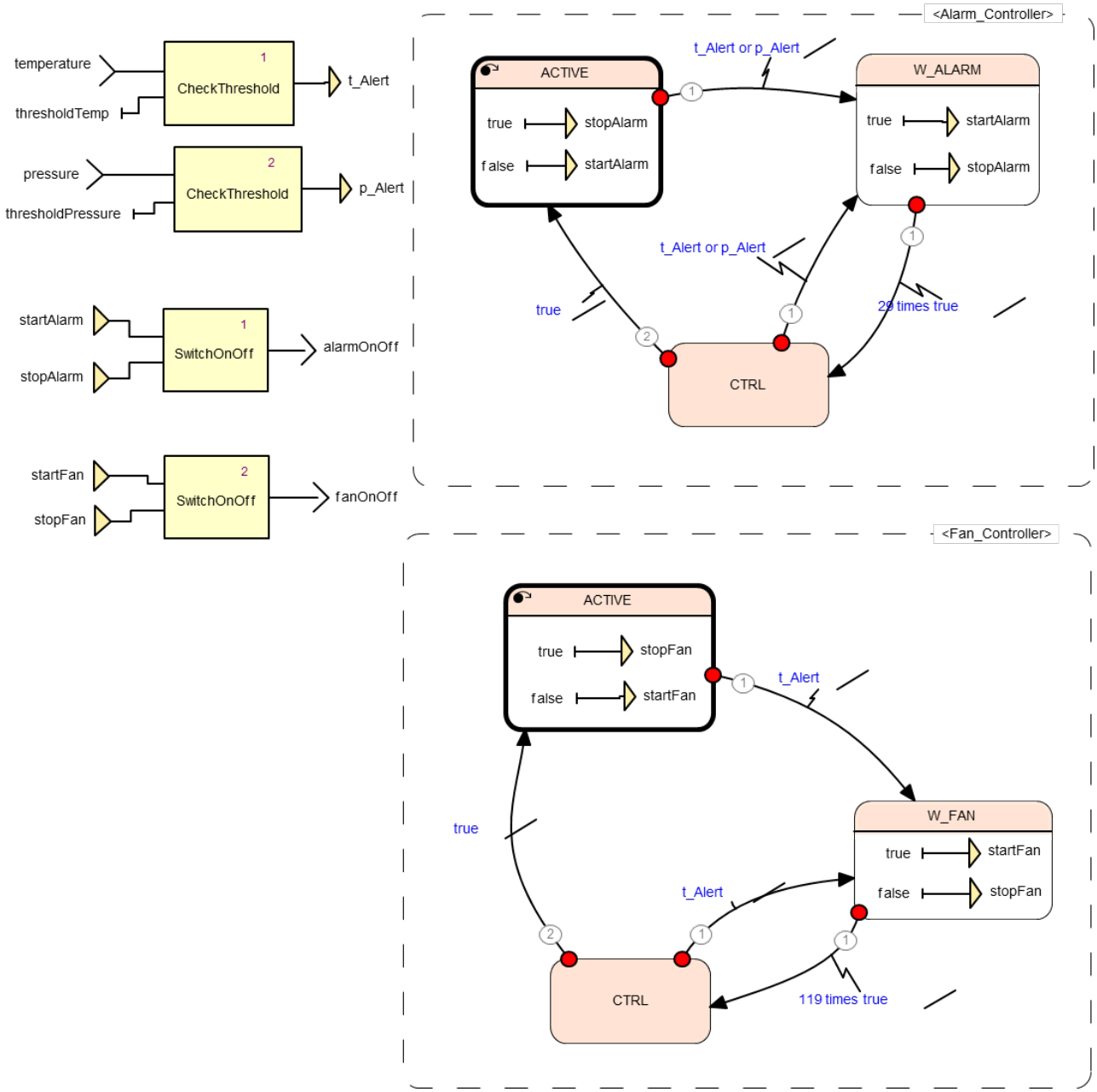


Figure 10: Implementation of the controller in Scade

5 Validation of the Model

First, a semantic verification of the model is performed using KCG code generator and simulation. Then, in order to validate, we run the following scenario shown in Fig.11:

- The temperature is 0.0 and the pressure is 0.0 for 10 cycles
- The temperature is 35.0 and the pressure is 0.0 for 220 cycles
- The temperature is 30.0 and the pressure is 0.0 for 60 cycles
- The temperature is 30.0 and the pressure is 15.0 for 130 cycles
- The temperature is 30.0 and the pressure is 10.0 for 160 cycles

- The temperature is 35.0 and the pressure is 15.0 for 165 cycles
- The temperature is 0.0 and the pressure is 0.0 for 95 cycles

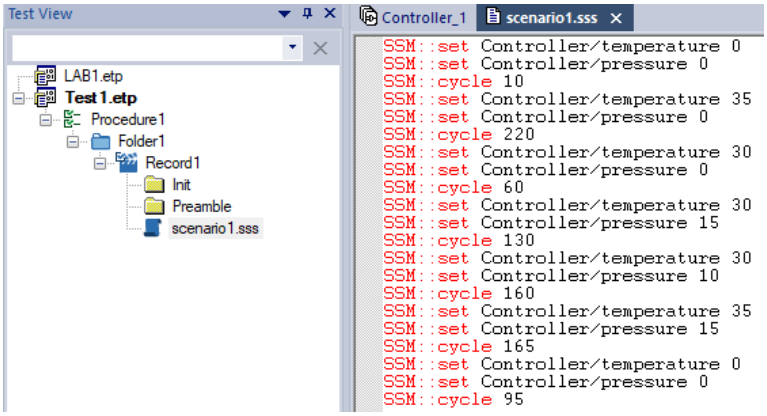


Figure 11: Simulation scenario definition

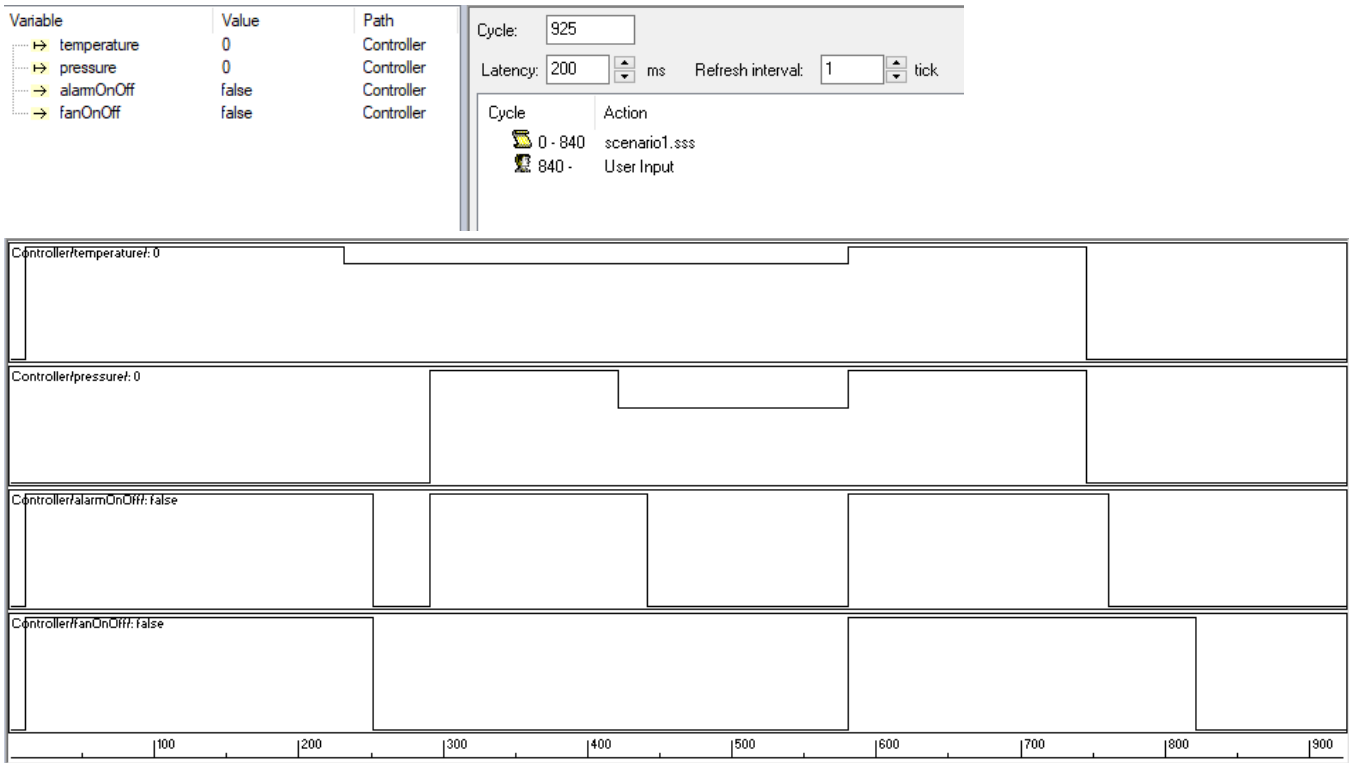
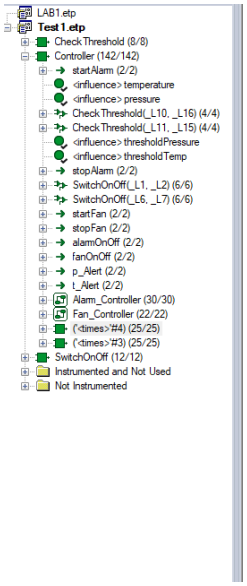


Figure 12: Observations of the simulation

The observations of the simulation are shown in Fig.12. We can see that the output signal alarmOnOff is true from the 10th to the 250th cycle triggered by the high temperature, from the 290th to the 440th cycle triggered by the high pressure, from the 580th to the 760th cycle triggered by both high pressure and high pressure. In the other cycles alarmOnOff is false. The output signal fanOnOff is true from the 10th to the 250th cycle and from the 580th to the 820th cycle both triggered by the high temperature. From the simulation results, we can observe that the requirements are satisfied.

6 Verification using Model Test Coverage

In order to test the coverage of our model, we simulate the scenario using the Model Test Coverage (MTC) tool. Fig.13 shows the coverage report of our model. We analyse the obtained coverage and add justification to reach a 100% coverage in the end.



Model Coverage summary

Date: 2021-12-05T22:23:40
Project: H:\scade\lab1\LAB1\LAB1.etp
Root Operators: Controller
Criterion: ODC

Test information:	Acquisition date	2021-12-05T22:21:42
	Test file	H:\scade\lab1\LAB1\Testing\Procedure1.stp
	Test operator	Controller
	Coverage file	H:\scade\lab1\LAB1\Testing\Test_MC\Procedure1\Procedure1_coverage.info
	01 Record	Record1
	Description	Record1
	Scenarios	
	01	H:\scade\lab1\LAB1\Scenario\scenario1.sss

Justification files: H:\scade\lab1\LAB1\Testing\Test1.mcj

Textual report file: H:\scade\lab1\LAB1\Testing\MC_Merged_Results\Test1_report.txt
Covered or justified / Total: 162 / 162 (100.00 %)
Covered or justified / Coverable: 156 / 156 (100.00 %)

Operator	Observed		Not observed				Total	% of total	% of coverable
			Coverable		Not coverable				
	Justified	Not justified	Justified	Not justified	Justified	Not justified			
CheckThreshold	0	8	0	0	0	0	8	100.00 %	100.00 %
Controller	0	132	4	0	6	0	142	100.00 %	100.00 %
SwitchOnOff	0	12	0	0	0	0	12	100.00 %	100.00 %
Total	0	152	4	0	6	0	162	100.00 %	100.00 %

Figure 13: Coverage of the model using Model Test Coverage (MTC) tool

7 Verification via Proof

Observers can implement the properties. An observer has a single Boolean output which must remain true as long as the inputs from the observer satisfy the property.

Three properties are verified in the controller system:

- The alarm is triggered if the temperature exceeds the temperature threshold.
- The alarm is triggered if the pressure exceeds the pressure threshold.
- The fan is set to work if the temperature exceeds the temperature threshold.

To verify the above-mentioned properties, we construct three functions called Prop1, Prop2 and Prop3 (Fig.14-16) and three nodes called Obs_Prop1, Obs_Prop2 and Obs_Prop3 (Fig. 17-19), on which we call the SCADE formal verification. Fig.20 show the verification results that all observers are proved to be valid.

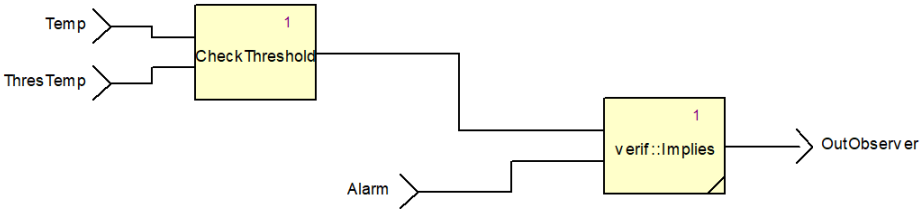


Figure 14: Prop1 function in Scade

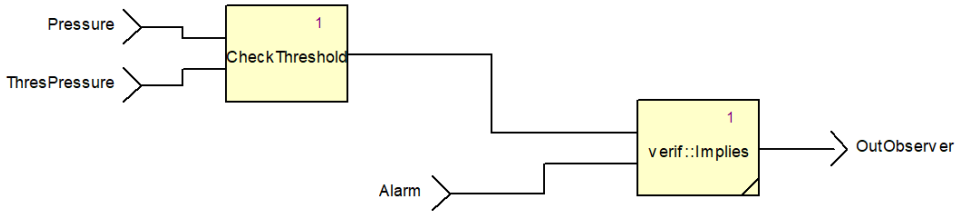


Figure 15: Prop2 function in Scade

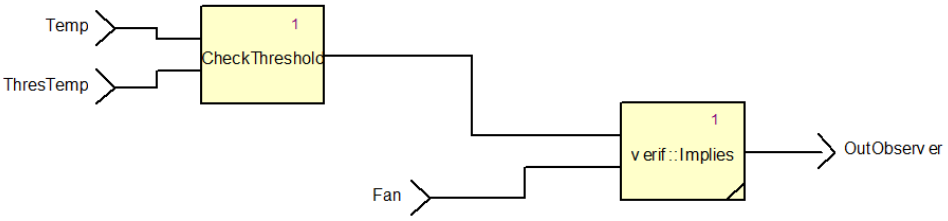


Figure 16: Prop3 function in Scade

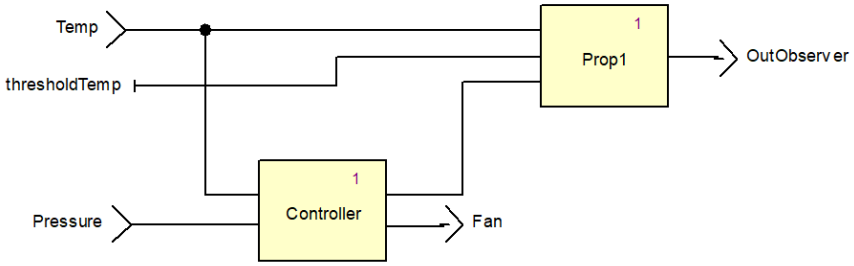


Figure 17: Obs_Prop1 node in Scade

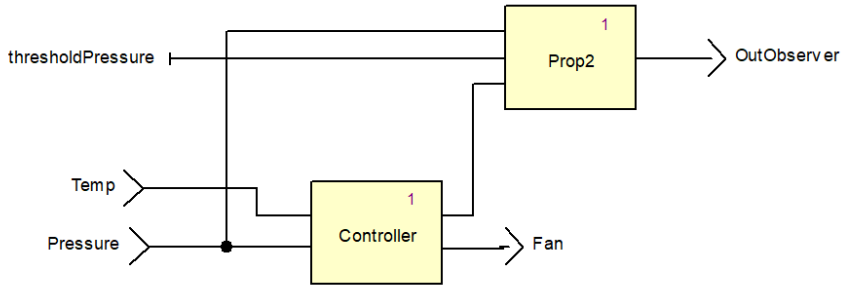


Figure 18: Obs_Prop2 node in Scade

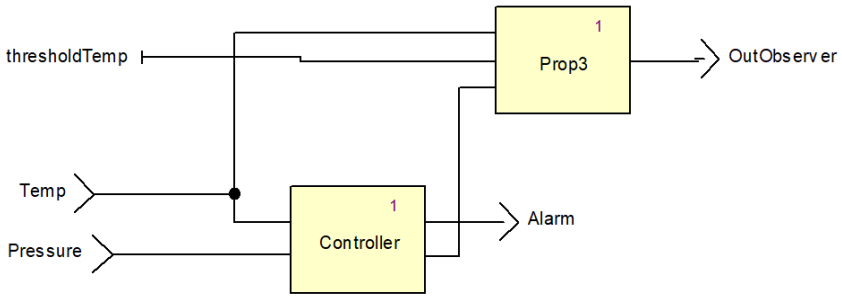


Figure 19: Obs_Prop3 node in Scade

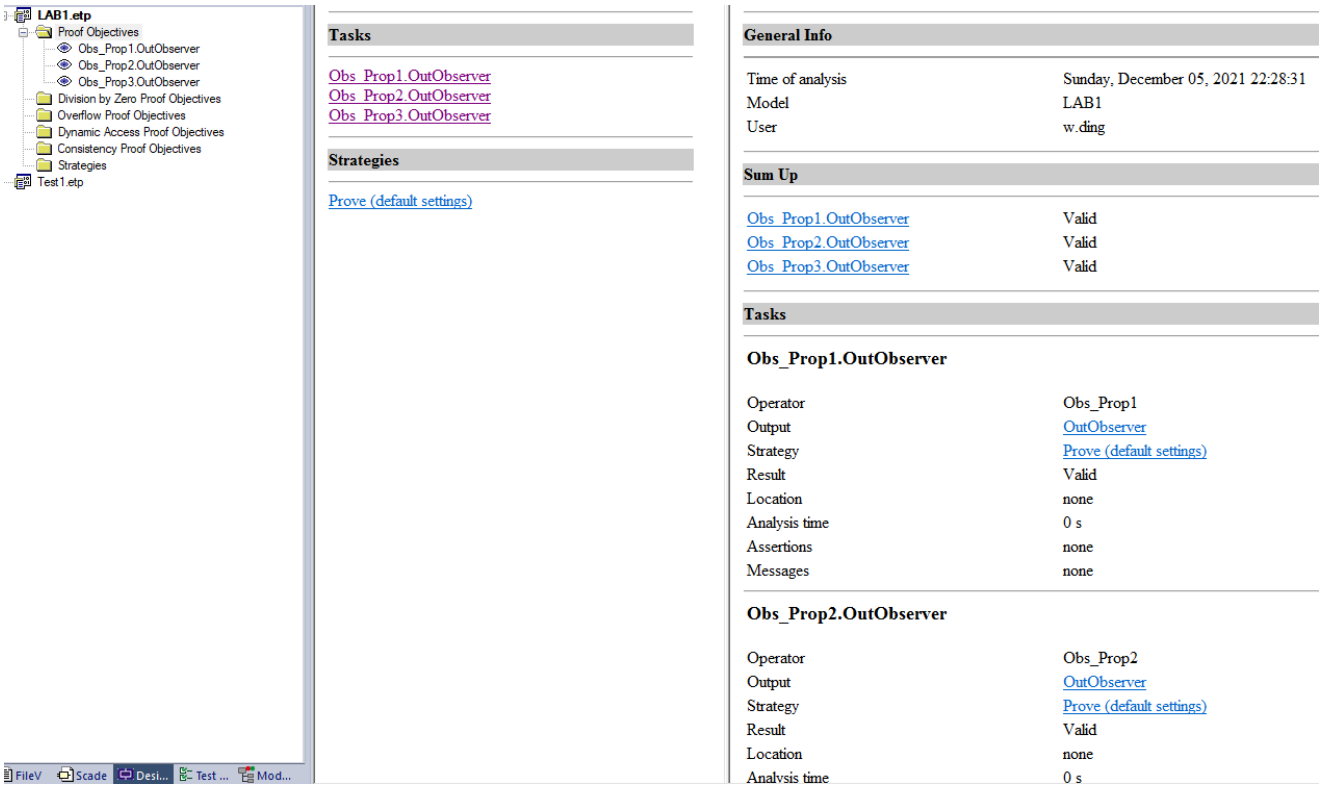


Figure 20: Verification via proof results in Scade

8 Integration on Card

In the integration phase the following elements are taken into consideration:

- use of a temperature sensor,
- exceeding from a fixed threshold in the SCADE model,
- triggering of an alarm,
- triggering of a fan.

Due to the limitation of equipment, we need to remove pressure sensor from the previous model. After removing all the functions concerned with pressure control, we only retain the temperature control functions (Fig.21) and do the validation and verification steps in Chapter 5-7 again.

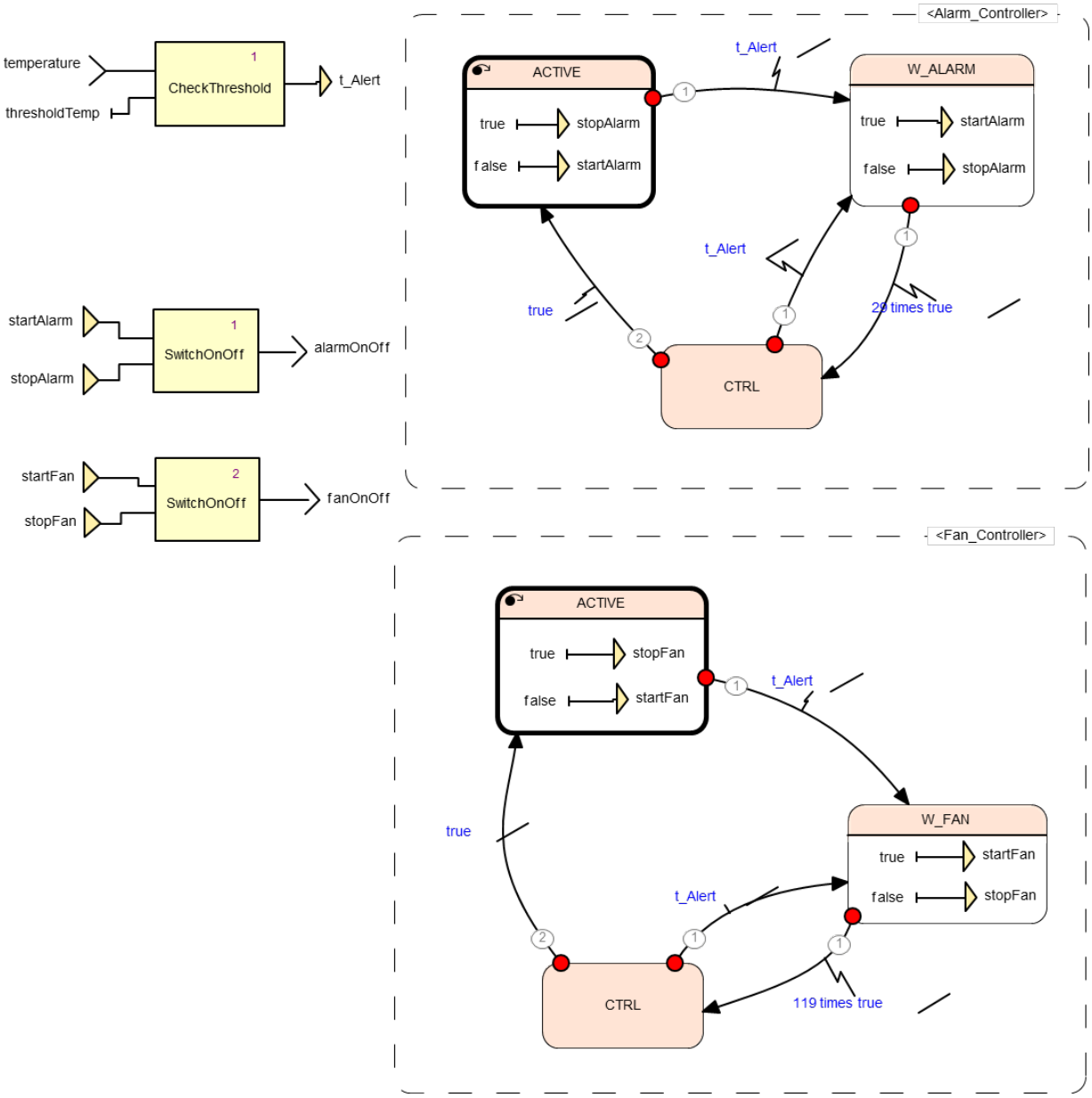


Figure 21: Controller model in Scade to be integrated on card

```
while (1) {
    /* Check if connected device is DS18B20 */
    if (TM_DS18B20_Is(DS_ROM)) {
        /* Everything is done */
        if (TM_DS18B20_AllDone(&OW)) {
            /* Read temperature from device */
            if (TM_DS18B20_Read(&OW, DS_ROM, &temp)) {
                /* Temp read OK, CRC is OK */

              /* evaluate SCADE operator */

                in.temperature = temp;

                Controller(&in, &out);

                  if (out.alarmOnOff == 1)
                  {
                    TM_DISCO_LedOn(LED_ORANGE);
                  }
                  else
                  {
                    TM_DISCO_LedOff(LED_ORANGE);
                  }

                  if (out.fanOnOff == 1)
                  {
                    TM_DISCO_LedOn(LED_BLUE);
                  }
                  else
                  {
                    TM_DISCO_LedOff(LED_BLUE);
                  }

                /* Start again on all sensors */
                TM_DS18B20_StartAll(&OW);

                /* Check temperature */
                if (temp > 30) {
                    TM_DISCO_LedOn(LED_RED);
                } else {
                    TM_DISCO_LedOff(LED_RED);
                }
            } else
            {
                /* CRC failed, hardware problems on data line */
            }
        }
    }
}
```

Figure 22: C code of main function of the controller model to be integrated on card

Then we generate the code and debug the errors, generate the bitstream and implement the code on a STM32 board which is connected to the Dallas DS18B20 digital temperature sensor. The modifications in the main function are shown in Fig.22. We set the blue light to simulate the fan operation and the red light to simulate the alarm ringing. The testing results are to our expectation, the functions are realized and the requirements are satisfied. The video of the testing process is accessible via the link: https://nextcloud.isae.fr/index.php/s/Ac3Ffnnfy7rHN3c.