ANTHONY DEBRUYN
BRIAN DELHAISSE
QUENTIN DELHAYE

Embedded Systems Design

The GOAT

The Gate Opener Actioned by Ticket

GILLES GERAERTS INFO-F-410

Contents

1 Introduction

For the course "Embedded Systems Design", we were asked to implement a system inspired from a subject in the course. We chose to use UppAal Tiga to design and create a strategy for a sas controlled by an Arduino Uno. This sas is composed of 2 doors actioned by servomotors. Infrared sensors are used to detect the person entering the sas. The opening of the second door of the sas is triggered by the scan of a valid card (rectangle with holes). Once someone is inside the sas, a timer is triggered to prevent the person to stay indefinitely inside. An alarm is activated once the timer is out.

2 The Model

In this section we will describe the physical properties that the system must satisfy, and make the link with the model.

First we want the system to always have at least one door closed. This is ensured by the 3 available configurations in the model: *01* meaning door 1 open and door 2 closed, *10* meaning the opposite, and *00* meaning no door open.

Secondly, we want the back door (door 2) to open only if the user present one valid card having 2 holes in it.

The user also has to be ejected after some time if he does not show a valid card. This would prevent the user from staying indefinitely in the sas, blocking the other users. This is done by the timer in the model for the sas and the eject message.

The sas must detect the insertion of a card, even if it is not valid. This is modeled with the cardIn message in the UppAal Tiga systems.

An infrared sensor system has to detect the entry and exit of the user in order to react with the appropriate door configuration, and (de)activate the alarm. In the model, the infrared sensors are modeled with the infra1 & infra2 signal.

The user must be able to escape the sas at any time. This is done with the infra1 sensor. If the user puts himself in front of the sensor, the front door opens to let him out.

The figures 1 and 2 represent these and the listings 1, 2 and 3 are their declarations.

A Typical run on the user side is as follows: The user begins in `before_sas` and proceed into the sas by triggering the `infra1` signal if the first door is opened. He enters either with a valid or invalid card. Once in the sas, he can present his card by triggering the `cardIn` signal. At any moment, if the second door is opened, he can leave through it, or leave through the first door by triggering the signal `infra1`. If the global clock `timeInSas` exceeds `SAS_TIMER`, he can go to the `FAIL` state, and when receiving the `eject` signal, he will leave the sas.

A typical run on the sas side is as follows: From the `idle` state, uppon `infra1` signal, the sas resets the clock `timeInSas` and closes the first door. The sas is then handling a user. When a detects that a card is entered, he either opens the second door if the card is correct but keep both closed if the card is invalid. If he receives the signal `infra1`, it means that user is standing in front of the first sensor and wants to leave, hence the sas opens the first door. Whenever the `timeInSas` clock reaches the `SAS_TIMER`, the sas immediatly triggers the `eject` signal, willing to make the user leave the bloody sas.

```
chan infra1, infra2, cardIn, eject;
clock timeInSas;
const int SAS_TIMER = 10;
int gateConf = 1;
int userCard;
```

Listing 1: Global declarations of the systems.

```
clock x;
```

Listing 2: Declarations of the user model.

```
clock x;
```

Listing 3: Declarations of the sas model.

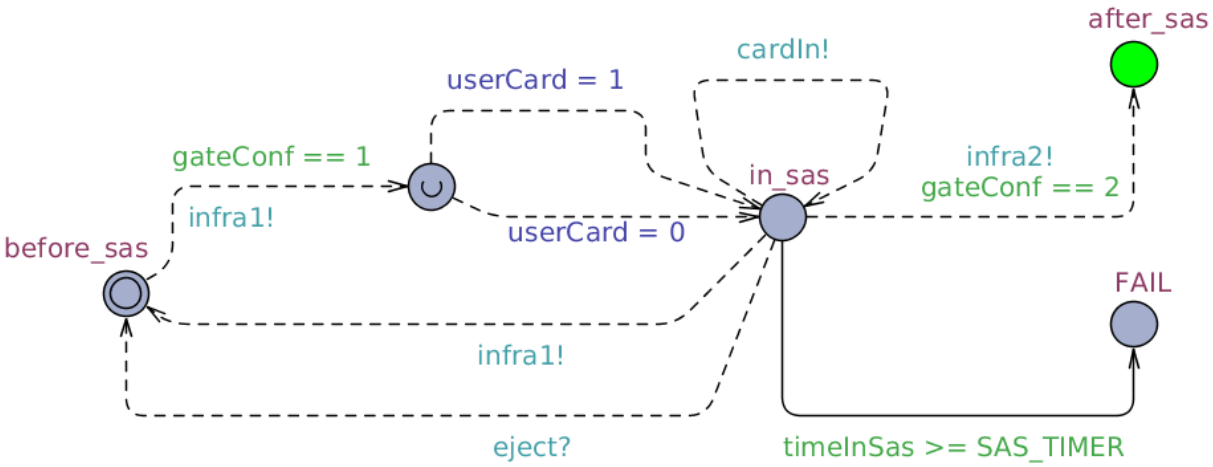

Figure 1: The user model

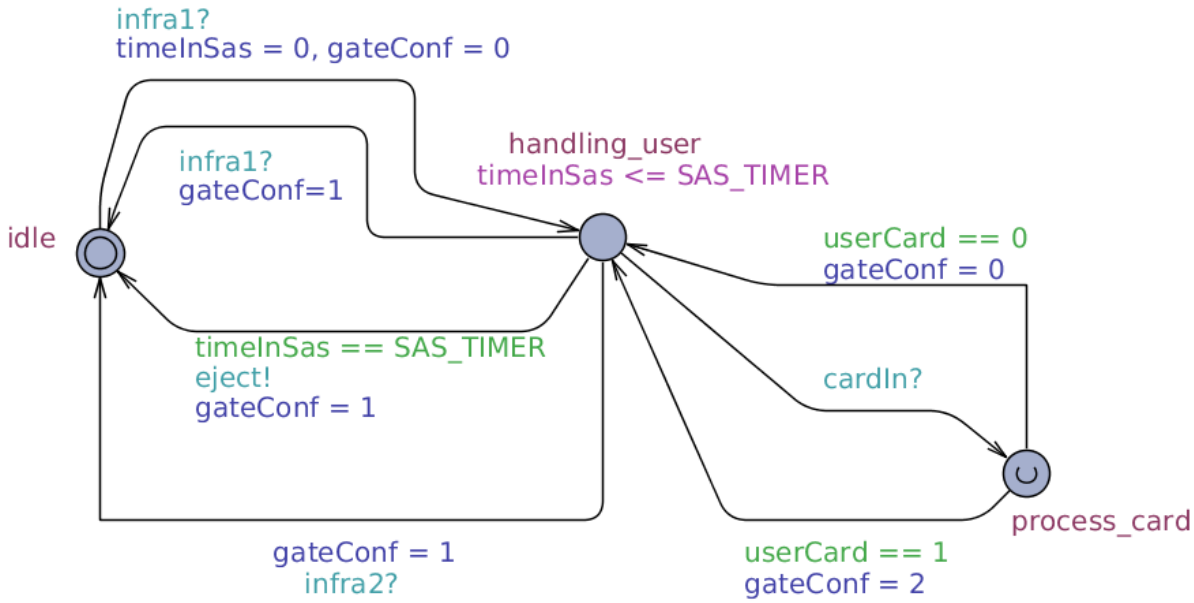

Figure 2: The sas model

3 The Strategy

3.1 Winning conditions

`A[] not(User.in_sas and User2.in_sas)`
Both users are never in the sas at the same time. Both models are exactly the same. The managment of a request token and a users queue will be implemented for the Formal Verification project.

`E<> User.after_sas`
It is possible for the user to reach the other side of the sas. He won't succeed on every try, but if he wants to, he will.

`A[] not (User.FAIL)`
We want to verify if the user will always avoid the FAIL state. At the instant `timeInSas = SAS_TIMER` the system has two choice: either take `Sas.handling_user` to `Sas.idle`, or `User.in_sas` to `User.before_sas`. The FAIL state is thus reachable and the test is expected to fail.

`control: A[] not(User.FAIL or User2.FAIL)`
Here, we play in a controller game. Having the controller collaboration, when the system is given

the opportunity to either win or lose, well, it chooses to win. This condition is there to enforce the limited time in sas property.

Please bear in mind here that we suppose that the `eject!` signal sent by the `Sas` has force of law. Hence, we know that when the controller is trying to comply with a winning condition, our physical system will be able to overcome the quite strong hypothesis made by the simulation tool stating that the environnement is always able to act faster than the controller. This liberty is represented by the controllable edge between `User.in_sas` and `user.FAIL`.

We may also want to verify that if the user enters the sas with an invalid card, he will not make it to the other side, but if he has a valid card, he may. This may be expressed as follows:

```
A[] ( ( (userCard == 0 && User.in_sas) imply A<> User.before_sas )
&& ( (userCard == 1 && User.in_sas) imply E<> User.after_sas ) )
```

Sadly, the syntax analyzer of UppAal does not accept a quantifier after the keyword `imply` and the above condition is not verifiable through its tools.

Finally, one could think that a suitable strategy for the sas to avoid the user to enter the `FAIL` state would simply be to never open the first door and let users in. However, the model construction is such that the first door is always open when the sas is not occupied. The only way the close the first door is a user to enter the sas by triggering the `infra1` signal.

3.2 Generated strategy

The following strategy has been generated using a system with only one user for readability purposes.

```
Verifying property 5 at line 28
 -- Property is satisfied.
$v_gameInfoPlayInitial state:
( User.before_sas Sas.idle ) gateConf=1
(timeInSas==User.x && User.x==Sas.x && Sas.x==0)

Strategy to avoid losing:

State: ( User.after_sas Sas.idle ) gateConf=1
While you are in true, wait.

State: ( User.before_sas Sas.idle ) gateConf=1
While you are in true, wait.

State: ( User.in_sas Sas.handling_user ) gateConf=2
While you are in (timeInSas<=10), wait.

State: ( User.in_sas Sas.process_card ) gateConf=0
When you are in (timeInSas<=10),
take transition Sas.process_card->Sas.handling_user { 1, tau, gateConf := 2 }

State: ( User.in_sas Sas.process_card ) gateConf=2
When you are in (timeInSas<=10),
take transition Sas.process_card->Sas.handling_user { 1, tau, gateConf := 2 }

State: ( User.in_sas Sas.handling_user ) gateConf=0
While you are in (timeInSas<=10), wait.
```

The implementation code is presented in appendix A. Each of those states were implemented using `if-then-else` statements decorated with all kind of computation that were hidden behind abstractions in the models.

4 Physical Implementation

4.1 Parts list

For our prototype, we used:

- a microcontroller: Arduino UNO.
- 5 resistances (220Ω).
- 3 leds (1red, 1yellow, 1green).
- 2 analog servo motors (FS5103B).
- 2 infrared proximity sensors short range.
- 2 photoresistors.
- 1 micro switch (DM3).
- 1 buzzer.
- 1 battery pack holder for 4x AA batteries + the 4xAA batteries.
- jump wires.
- 3 half-size protoboards.

4.2 Electronic schematic

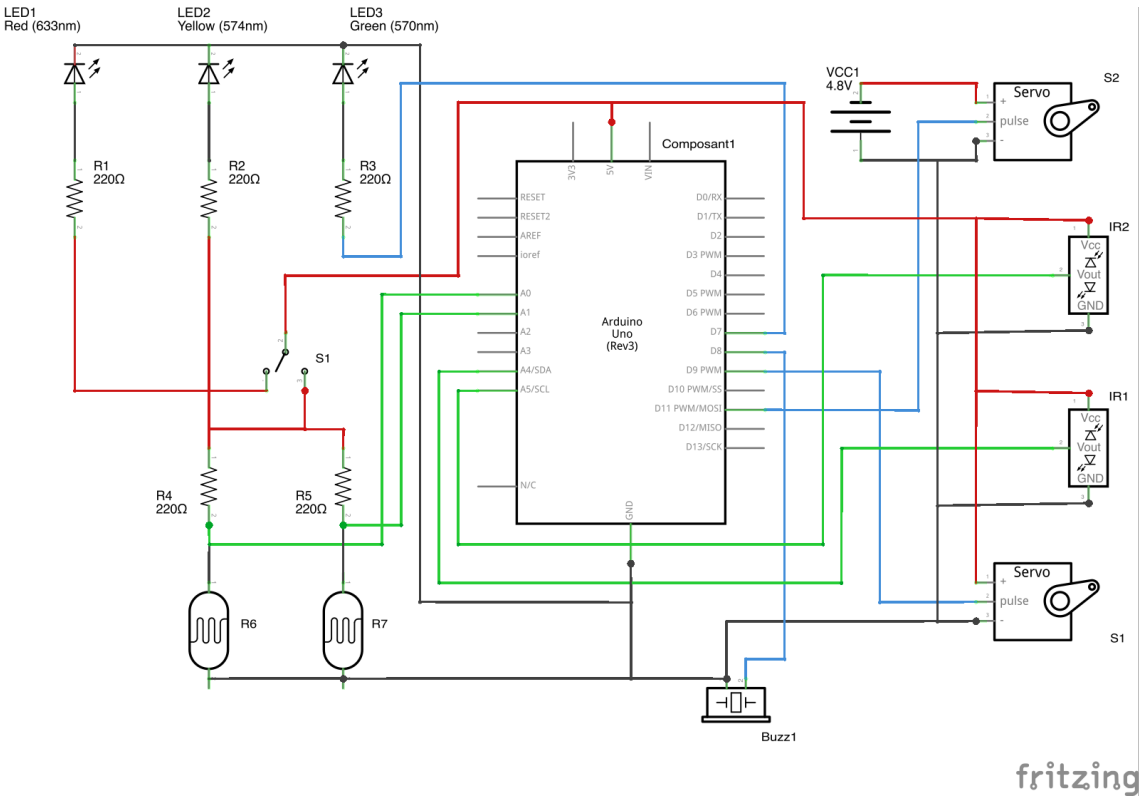

Figure 3: Electrical diagram (modeled with Fritzing). Legend: red wires are connected to the positive terminal (of the arduino or the external battery), black wires are connected to the ground, green wires connect the electrical components to the analog input pins, and the blue wires connect electrical components to digital output pins.

Once the arduino is connected to a computer or to a battery, the positive terminal of the arduino supplies a tension of 5V to the electrical system, which can be seen on the figure 3. Because the servo

motors drain a lot of current, another external battery was needed to supply enough current for both servo motors. Thus, an external battery has been connected to one of the servo. To avoid any other problems, we connected the ground of this one to the same ground as the arduino.

The resistors in front of each led are there so that the leds don't overdrive and burn out. The red led is always on until a card is inserted into the system. The card will push on the lever of the micro switch leading to the yellow led to turn on, and to the current to flow into the photoresistors.

The 2 resistors R4 and R5 with the photoresistors R6 and R7 form respectively a voltage divider. The voltage measured differs with the intensity of the light, that is, brighter the light is smaller the resistance of the photoresistor is. The resulting tension of those dividers are measured on the analog input pins A0 and A1 of the arduino. If the tension of each photoresistor is between a certain value, then the microcontroller is programmed to turn on the green led, and activate the 2nd servo motor (S2) in order to open the last door of the sas.

If the user stays too long in the sas, the arduino can also trigger an alarm with the buzzer.

The two infrared proximity sensors IR1 and IR2 are there to detect if an object/person has entered or leaved the sas, they are respectively connected to the analog input pins A4 and A5. Based on the value measured, the microcontroller will activate one of the servo motors to close one of the gates, or to stop the buzzer.

4.3 Electronic circuit

On the real circuit, we used three breadboards because of the jump wires length that were too short. The two leftern protoboards on the figure 4 are perpendicular to each other, in order for the card to be able to push on the lever of the micro switch, and for the photoresistors to be perpendicular to the card.

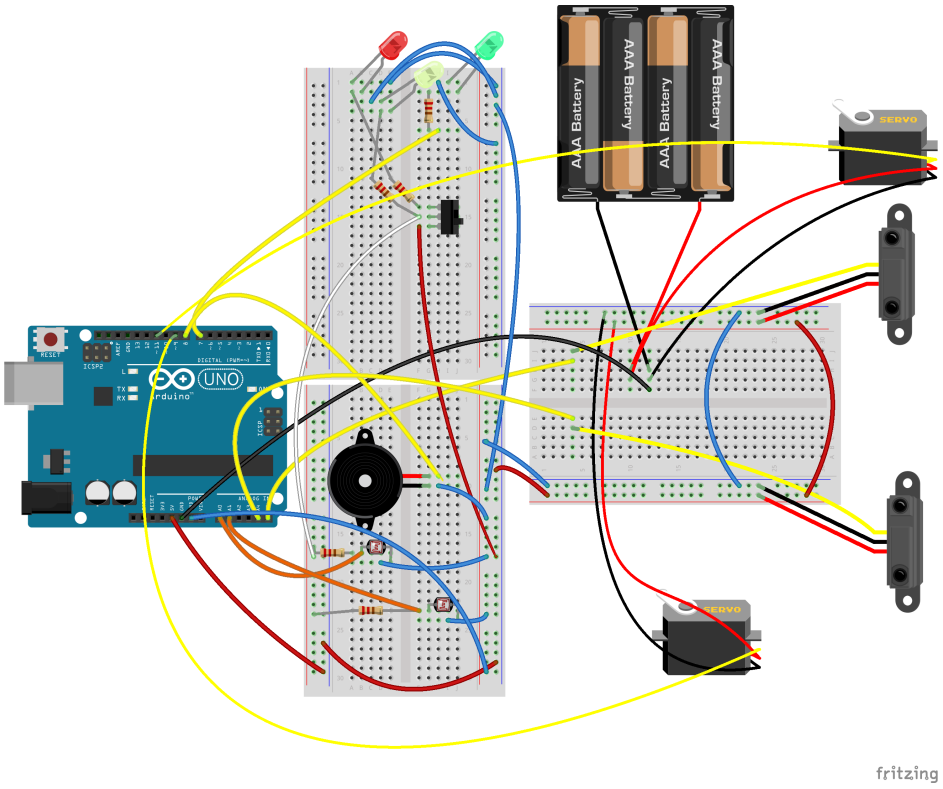

Figure 4: The Circuit (modeled with Fritzing).

4.4 The structure

The sas was build using Lego bricks, and the doors are in carton and connected to the servomotors with paperclips. We used cardboard to create some cards. A valid card has a hole in front of each photoresistor, so that the light can pass through it and be detected by the photoresistors. The final physical prototype can be found on the figure 5.

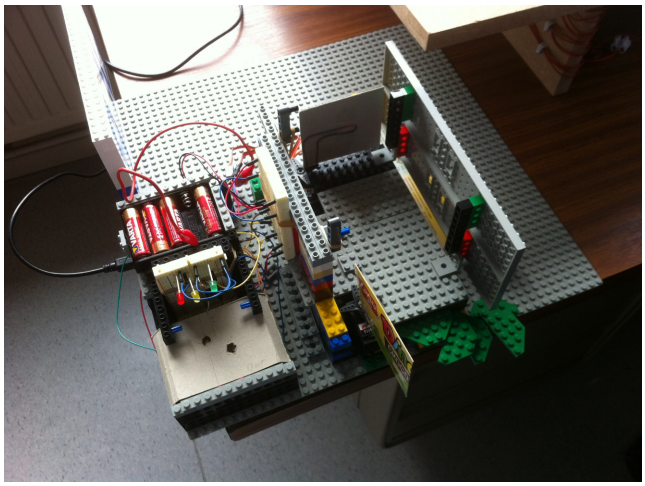

Figure 5: Final prototype

5 Future work

Since we only use one infrared sensor at each door, we do not know the direction of the passing user. This causes problems in several situations:

- If the user stays too long in the sas and the alarm activate, another user can enter in the sas. The alarm then stops. Now a third user can enter, and the front door closes. If one of the 2 users already in the sas puts something in front of the sensor, the front door closes as well. Thus any number of people enter the sas and go to the other side with one valid card.
- The same infrared problem occurs on the other side. If one user enters the sas from the back door when it is open, after the user has shown a valid card, the sas closes the second door. There are then 2 users in the sas, and the front door opens again, letting the second user escape.

These two problems could be resolved by using 2 more sensors to detect the direction of the passing user. Anoher solution could be to consider that the sas has only room for one person only.

6 Conclusion

This project has given us the opportunity to understand how we design embedded systems, by using specific tools to verify that a system respect some given proprieties. It was a rich experience to do the different parts of a project: the design of the model, the production of the code from the generated strategies, and the physical implementation so that we can see that our model really works in the real world.

A Arduino Code

As mentionned above, the following code has been produced based on the generated strategies.

```
#include <Servo.h>

Servo servo1; // create servo object to control a servo
Servo servo2;

int pos1;   // variable to store the servo position
int pos2;

const int ledPin = 7;    // the number of the LED pin
const int buz = 8;
int buzCount = 0;

int gateConf;
unsigned long startTime;
const int SAS_TIMER = 5000;
boolean userIn;

#define but1 A4
#define but2 A5
#define infra1 (analogRead(A4) > 550)
int infra1Count;
#define infra2 (analogRead(A5) > 500)
int infra2Count;
#define cardIn (analogRead(A0) > 0 && analogRead(A1) > 0)

void setup()
{
  pos1 = 0;//Opened
  pos2 = 90;//Closed
  servo1.attach(9); // attaches the servo on pin 9 to the servo object
  servo2.attach(11);
  Serial.begin(9600);
  servo1.write(pos1);
  delay(150);
  servo2.write(pos2);
  delay(150);

  // set the digital pin as output:
  pinMode(ledPin, OUTPUT);
  pinMode(buz, OUTPUT);

  gateConf = 1;
  userIn = false;
  infra1Count = 0;
  infra2Count = 0;
}

void changeGateConf(int conf)
{
  gateConf = conf;
  switch (conf) {
    case 0:
      //from 1
      //Close gate 1
      for(pos1; pos1 < 90; pos1 += 1) {
        servo1.write(pos1);
        delay(15);
      }
      //gate 2 already closed
      break;
```

```
61
62      case 1:
63        //Open gate 1 : 0->90
64        //Close gate 2 : 180->90
65        for(pos2; pos2 > 90; pos2 -= 1) {
66          servo2.write(pos2);
67          delay(15);
68        }
69        for(pos1; pos1 > 0; pos1 -= 1) {
70          servo1.write(pos1);
71          delay(15);
72        }
73        break;
74
75      case 2:
76        //gateConf = 2
77        //Close gate 1: 90->0
78        //Open gate 2: 0->90
79
80        for(pos1; pos1 < 90; pos1 += 1) {
81          servo1.write(pos1);
82          delay(15);
83        }
84        for(pos2; pos2 < 180; pos2 += 1) {
85          servo2.write(pos2);
86          delay(15);
87        }
88        break;
89
90      default:
91         break;
92    }
93  }
94
95  boolean processCard()
96  {
97    int sensorValue = analogRead(A0);
98    int sensorValue2 = analogRead(A1);
99    if(sensorValue>200 && sensorValue<400 && sensorValue2>200 && sensorValue2<400)
100     return true;
101   else
102     return false;
103 }
104
105 void alarm(boolean on)
106 {
107   if(on) {
108     int i;
109     if(buzCount < 9) {
110       for(i=0; i<10; i++) {
111         digitalWrite(buz, HIGH);
112         delay(3);//Shorter delay = higher pitch
113         digitalWrite(buz, LOW);
114         delay(3);
115       }
116       buzCount ++;
117     } else {
118       for(i=0; i<10; i++) {
119         digitalWrite(buz, HIGH);
120         delay(18);
121         digitalWrite(buz, LOW);
122         delay(18);
123       }
124       buzCount = 0;
```

```
    }
  }
}

void eject()
{
  digitalWrite(ledPin, LOW);
  changeGateConf(1);
  Serial.println("GET OUT. NOW");
  while(!infra1) {
    Serial.println("GET OUT. NOW");
    alarm(true);
  }
  while(infra1);
  alarm(false);
}

void loop()
{
  //infra1!
  if(infra1 && gateConf == 1 && !userIn) {
    userIn = true;
    Serial.print("userIn: ");
    Serial.println(userIn);
    gateConf = 0;
    changeGateConf(gateConf);
    startTime = millis();
  }

  if(infra2 && gateConf == 2) {
    gateConf = 1;
    userIn = false;
    digitalWrite(ledPin, LOW);
    changeGateConf(gateConf);
  }

  if(gateConf == 0 || gateConf == 2) {
    if((millis() - startTime) <= SAS_TIMER) {
      if(cardIn) {
        if(processCard()) {
          digitalWrite(ledPin, HIGH);
          gateConf = 2;
          changeGateConf(gateConf);
        }
        else
          digitalWrite(ledPin, LOW);
      }
      else
        digitalWrite(ledPin, LOW);
    }
    else {
      eject();
      userIn = false;
    }
  }

  delay(50);  // delay in between reads for stability
}
```