METHODOLOGIES FOR DISCRETE EVENT MODELING & SIMULATION

CARLETON UNIVERSITY



Victor Soto Garcia

Vsoto101@uottawa.ca

# Project Model Definition

This project will focus on modeling the behavior of an underground parking lot. For this purpose, a one level parking lot with 50 spaces divided on 2 sides will be used. The parking lot will have sensors on top of each space so the system will know how many spaces are available; there will also be a screen at specific points in the parking lot indicating the number and the direction of available spaces. This so the drivers can find an empty parking spot faster and reduce queue time in parking lots.

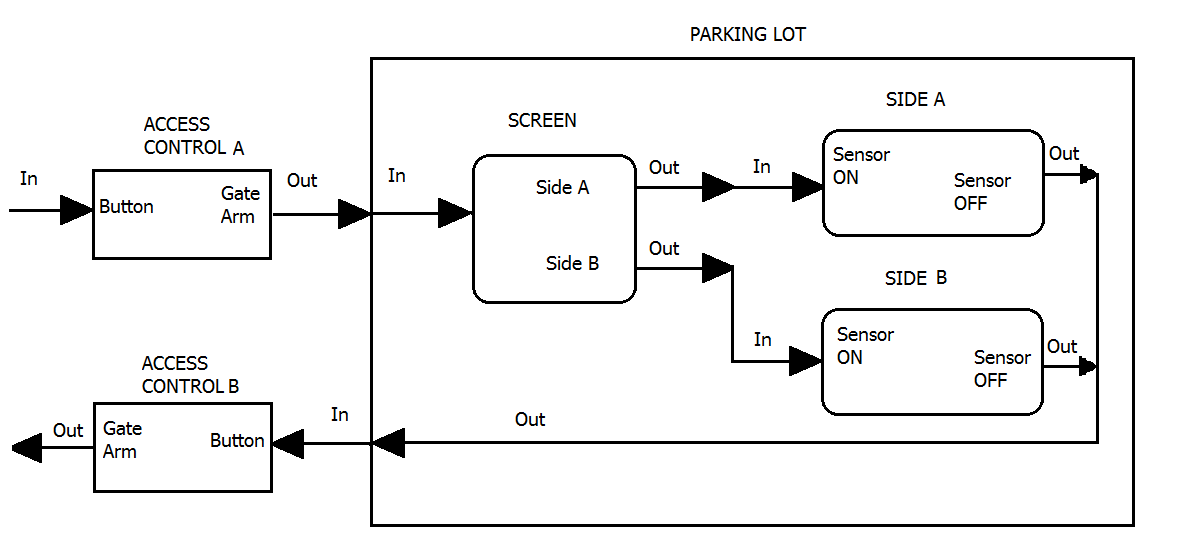


Figure 1 - Model Structure

* **Components**

- **Access control A**: this component has a random input and can generate an output when the “Button” function is activated.

- **Screen**: receives an input and generates an output depending on what function is activated, “Side A” or “Side B”.

- **Side A**: receives an input from “Screen” and generates an output once the function “Sensor OFF” is activated

- **Side B**: receives an input from “Screen” and generates an output once the function “Sensor OFF” is activated

- **Access control B**: receives an input from “Parking Lot” and generates an output once the function “Gate arm” is activated

- **Parking Lot**: Receives an input from “Access control A” and is formed by 3 other submodels

To understand better its behavior we can see use an example with an empty parking lot following Figure 1. The car will go into the access control, press the button for the gate arm to open and then go inside the parking lot, where the driver in the car will look into the screen and see wich side has an available space, side A or side B, assuming the car goes side B then it will take a parking spot wich will activate a sensor and subtract a space in the screen leaving 24 spaces available. When the car leaves the sensor will turn off and the screen will add one empty space to side B and there will be 25 again.

# Models Formal Specifications

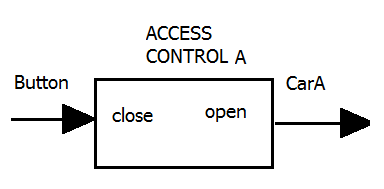
As shown in Figure 1 the Parking Lot has 1 input and 1 output. Input is conditioned by “Acces Gate A” wich generates an output once it is activated, output generated from Parking Lot becomes an input for “Acces Gate B” that once is activated it allows the car to leave the building. The Parking Lot is formed by 3 components: Screen, Side A and Side B. Screen sends cars either to Side A or Side B and then the side chosen activates Sensor ON for a certain period of time, once this period is finished it activates Sensor OFF to generate an output for Parking Lot.

**Atomic Models**

The formal specifications <S, X, Y, δint, δext, λ, ta> for the ***atomic models*** are defined as follows:

* **Access Control A**

Access Control is a passive atomic model. It reacts to the input of pressing a button (1 sec), and outputs a value carA simulating a car arriving.

****

S = {close, open}

X = {button}

Y = {carA}

δint (close) = passive

δext (close, e, button)

{ if spacesB>0;

Open;

else:

passivate; //parking lot is full

}

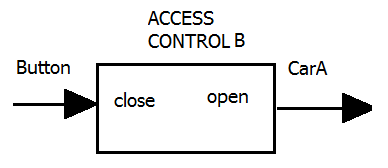
λ(open) = car

ta(close) = INFINITY

ta(open) = pressButton

* **Access Control B**

Access Control is a passive atomic model. It reacts to the input of pressing a button (1 sec), and outputs a value carL simulating a car leaving.



S = {close, open}

X = {button}

Y = {carA}

δint (close) = passive

δext (close, e, button)

{ if button=1;

Open gate;

else;

Passivate;

}

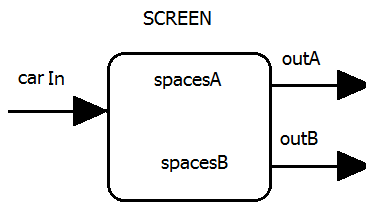
λ(open) = car

ta(close) = INFINITY

ta(open) = pressButton

* **Screen**

Screen is an active atomic model. It is always showing the number of parking spaces available and if they are at Side A or B, the outputs outA and outB simulates a car choosing a side.

****

State Variables:

spacesA = 25; // number of total car spaces on sideA

spacesB = 25; // number of total car spaces on sideB

S = {spacesA, spacesB}

X = {carIn}

Y = {outA, outB}

δint (active) = active

δext (spacesA, spacesB, e, carIn) = active

{ case carIn

spacesA:

if spacesA=0;

goto spacesB;

else:

output=outA

spacesB:

if spacesB>1

output=outB

else:

//shouldn’t exist, no spaces available.

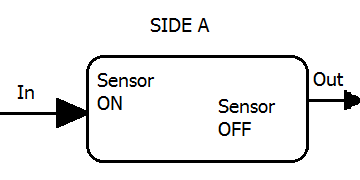
}

λ(active) = OutA, OutB

ta(active) = INFINITY

* **Side A**

Side A is an active atomic model. It reacts to the input of a car being sent to a specific side of the parking lot by “Screen”, it has a sensor on top of each parking space with a light to indicate it is empty and turns off when a car parks there. It outputs a value Out simulating a car leaving.

****

State variable = available =25

S = {On,Off}

X = {In}

Y = {Out}

δint (ON) = active

δext (In, e, )

{

If Input

{

Sensor Off;

Available=available-1;

}

}

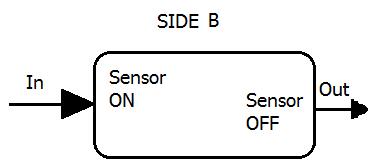
λ(open) = car

ta(ON) = INFINITY

ta(OFF)= Random from 1 to 60 minutes

* **Side B**

Side A is an active atomic model. It reacts to the input of a car being sent to a specific side of the parking lot by “Screen”, it has a sensor on top of each parking space with a light to indicate it is empty and turns off when a car parks there. It outputs a value Out simulating a car leaving.

****

State variable = available =25

S = {On,Off}

X = {In}

Y = {Out}

δint (ON) = active

δext (In, e, )

{

If Input

{

Sensor Off;

Available=available-1;

}

}

λ(open) = car

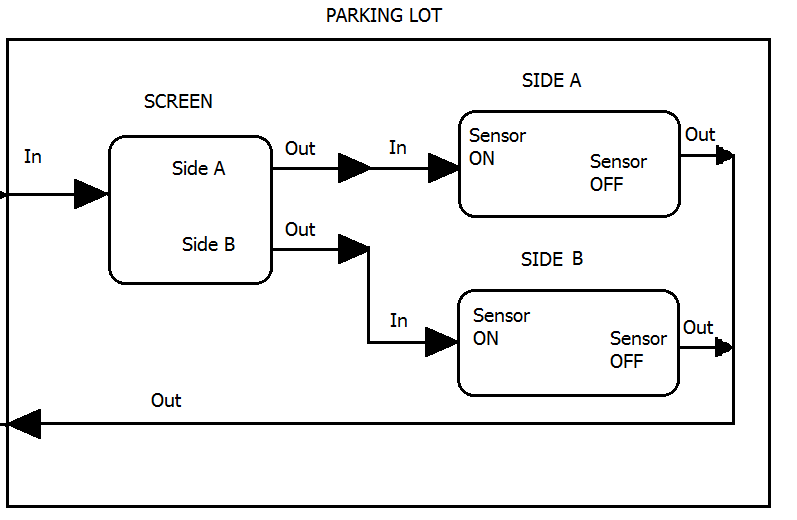
ta(ON) = INFINITY

ta(OFF)= Random from 1 to 60 minutes

**Coupled Models**

The formal specifications <X, Y, D, EIC, EOC, IC, SELECT > for the ***coupled model*** ParkingLot is defined as follows:

* **Parking Lot**

****

X = {carA};

Y = {carL};

D = {Screen, SideA, SideB};

EIC = {(accessControlA.carA, Screen.in)}

EOC = {(sideA.out, carL), (sideB.out, carL)}

IC = {(Screen.out, sideA.in), (Screen.out, sideB.in)

SELECT = Screen

**Test Strategies**

The atomic models and coupled models will be tested using an event file. Test cases are created by adding different combinations of inputs to the “ParkingLot.ev” file, running the simulation and checking whether the outputs in the output and log files are what we expected.

# CD++ Simulation

For the simulation using CD++ a simpler model was used.

Access Control atomic models

Basically this model is queue and its only time advance is 5 seconds from the gate opening wait, the event file was created as follows to check the delay was working.

*00:00:00:00 in 1*

*00:00:06:00 in 1*

*00:00:13:00 in 1*

00:00:19:00 in 1

In the output file *AccessControlA.out ,* the results came out correct.

*00:00:05:00 out 1*

*00:00:11:00 out 1*

*00:00:18:00 out 1*

00:00:24:00 out 1

**Coupled Model ParkingLot:**

For this model instead of using SideA and SideB, a Sensor1 model was used to simulate the decreasing value of the parking spaces, along with a Screen atomic model. The event file was created as follows

00:00:00:00 in 1

00:00:10:00 in 1

00:00:20:00 in 1

00:00:30:00 in 1

To check the results a starting value of 50 was used for the spaces available in the Parking Lot

00:00:01:000 out 1

00:00:01:000 out 49

00:00:07:000 out 1

00:00:07:000 out 48

00:00:17:000 out 1

00:00:17:000 out 47

00:00:27:000 out 1

00:00:27:000 out 46

00:00:37:000 out 1

00:00:37:000 out 45

**Conclusion**

The simpler model works as expected, it counts the spaces that has been used. The future work for the CD++ simulation is to add more sides and a function to the Screen model that lets you know how many spaces are on each side.