**Report**

According to the map given to each team, develop a controller for each intersection (plant), that controller is a closed-loop one (with the *in(1..n)*input channels that is connected to its intersection’s output channels *op(1..n)* and an Intersections (with the *OPs*output channels). The controller must have dynamic delays feature to extend the time of the green light in case of a traffic jam. (Project session 3, and 5)

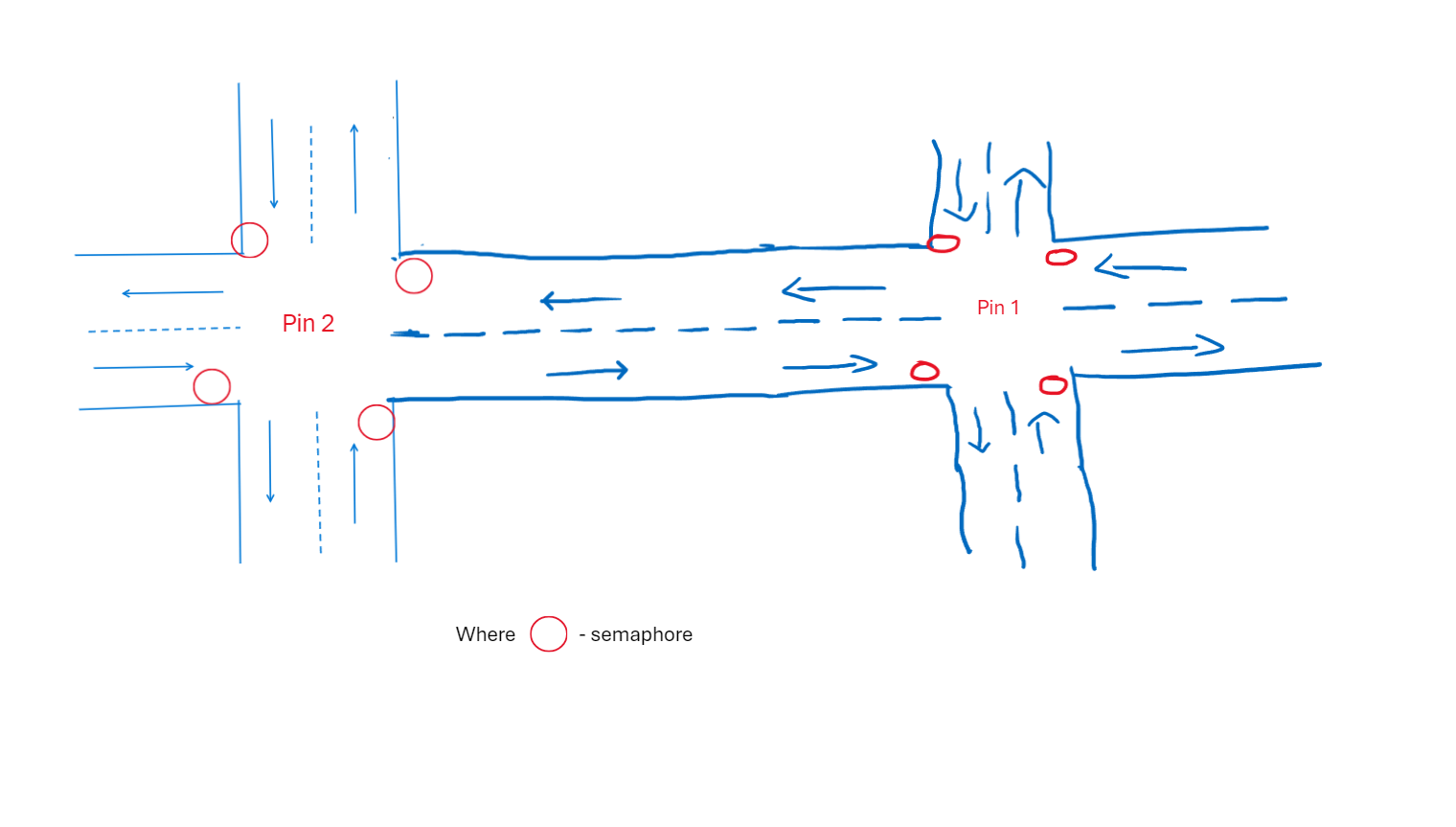
**Specifications:**

1. Screen shot of the entire given map,

O imagine care conține hartă, text, atlas

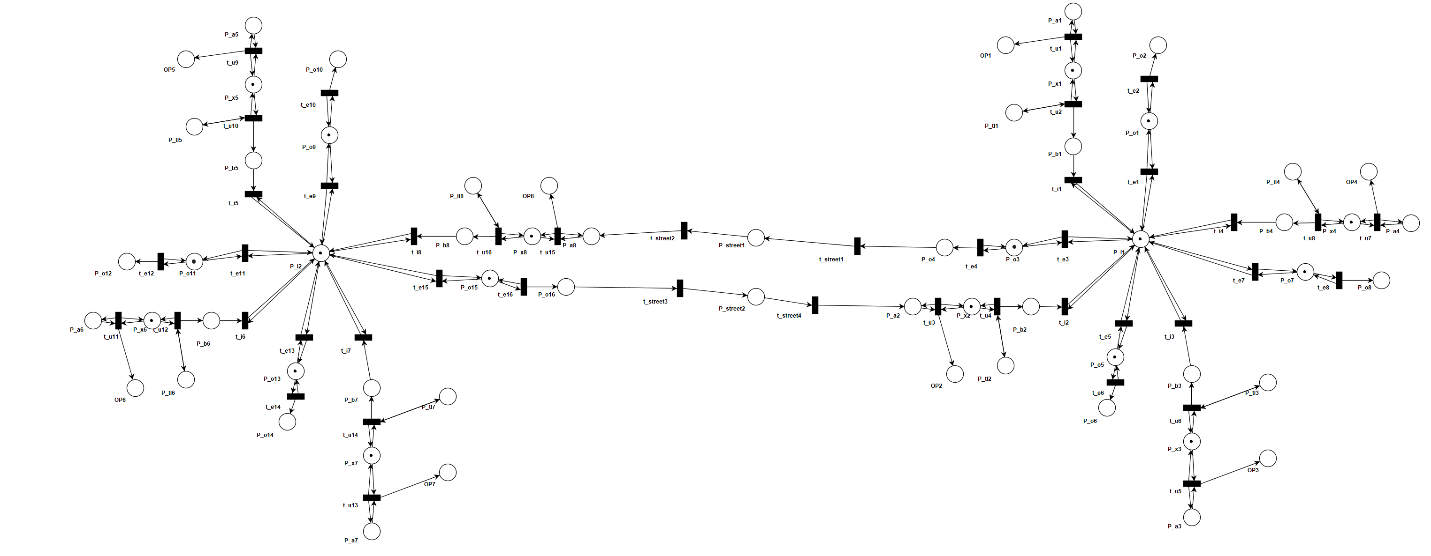
Descriere generată automat

1. Simplified one showing the intersections and the middle street that connects them, if the street has output and input lanes, they should be drawn and implemented at the end.



**Design:**

1. The OETPN model for the Plant (the intersections and the middle street).



1. Place types, grd&map for the entire OETPN.

**Place Types**P\_a1, P\_a2, P\_a3, P\_a4, P\_a5, P\_a6, P\_a7, P\_a8, P\_b1, P\_b2, P\_b3, P\_b4, P\_b5 P\_b6, P\_b7, P\_b8, P\_o2,P\_o4, P\_o6, P\_o8, P\_o10, P\_o12, P\_o14, P\_o16, P\_street1, P\_street2- **Data Car**

P\_x1, P\_x2, P\_x3, P\_x4, P\_x5, P\_x6, P\_x7, P\_x8, P\_I1, P\_I2, P\_o1, P\_o3, P\_o5, P\_o7, P\_o9, P\_o11, P\_o13, P\_o15 – **Data Car Queue**

P\_tl1, P\_tl2, P\_tl3, P\_tl4, P\_tl5, P\_tl6, P\_tl7, P\_tl8 – **Data String**

OP1, OP2, OP3, OP4, OP5, OP6, OP7, OP8 – **Data Transfer**

**Grd&map**

**t\_u1** : P\_a1 != null AND P\_x1.CanAddCars => P\_x1.AddElement(P\_a1)

P\_a1 != null AND P\_x1.CanNotAddCars => OP1.SendOverNetwork(“Full”)   
 => P.a1=P.a1

\*same for **t\_u3**, **t\_u5**, **t\_u7**, **t\_u9**, **t\_u11**, **t\_u13**, **t\_u15**\*

**t\_u2:** P\_x1.HaveCar AND P\_tl1= “green” => P\_x1.PopElementWithoutTarget(P\_b1)

=> P\_tl1=P\_tl1

\*same for **t\_u4**, **t\_u6**, **t\_u8**, **t\_u10**, **t\_u12**, **t\_u14**, **t\_u16**\*

**t\_i1:** P\_b1 != null AND P\_I1.CanAddCarrs => P\_I1.AddElement(P\_b1)

\*same for **t\_i2, t\_i3, t\_i4, t\_i5, t\_i6, t\_i7, t\_i8**\*

**t\_e1:** P\_o1.CanAddCars AND P\_I1.HaveCarForMe => P\_I1.PopElementWithTargetToQueue(P\_o1)

\*same for **t\_e3, t\_e5, t\_e7, t\_e9, t\_e11, t\_e13, t\_e15**\*

**t\_e2:** P\_o1.HaveCar => P\_o1.PopElementWithoutTarget(P\_o2)

\*same for **t\_e4, t\_e6, t\_e8, t\_e10, t\_e12, t\_e14, t\_e16**\*

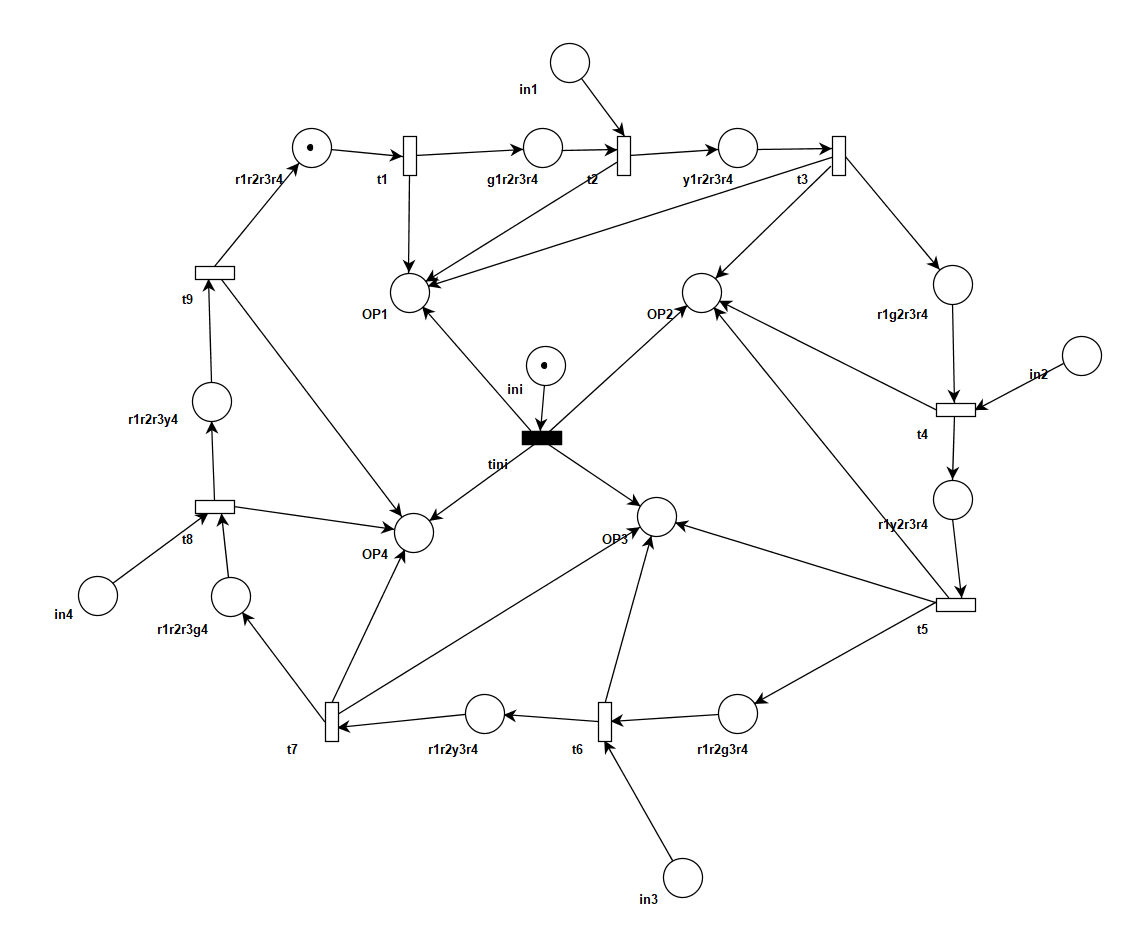
**t\_street1:** P\_o4 !=null => P\_street1=P\_o4

\*same for **t\_street3**\*

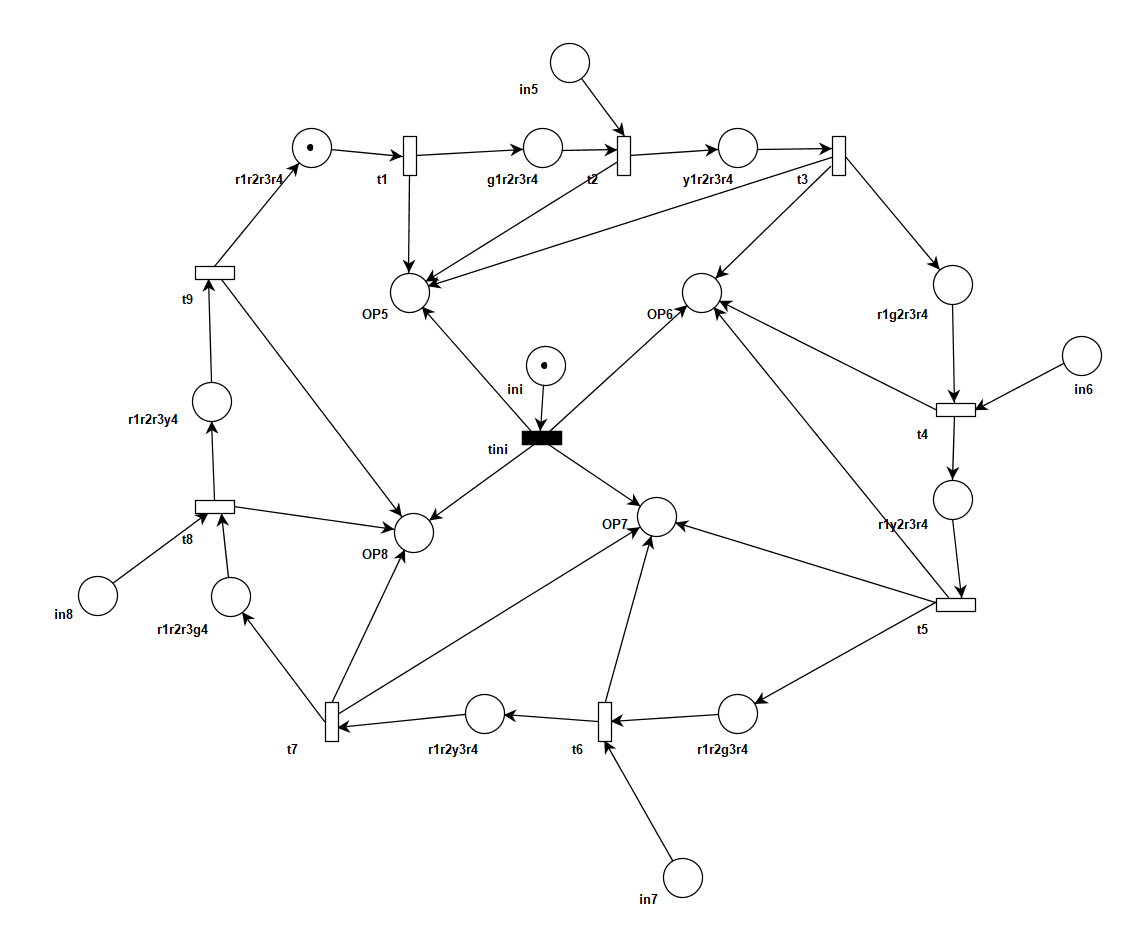
**t\_street2:** P\_street1!=null => P\_a8=P\_street1

\*same for **t\_street4**\*

1. The OETPN model for the controllers.



Controller 1



Controller 2

1. The Place types, grd&map for the entire OETPN.

**Controller 1**

**Place Types**

Ini, R1r2r3r4, g1r2r3r4, y1r2r3r4, r1g2r3r4, r1y2r3r4, r1r2g3r4, r1r2y3r4, r1r2r3g4, r1r2r3y4– Data String

OP1, OP2, OP3, OP4, in1, in2, in3, in4 – Data Transfer

**Grd&map**

iniT : ini!= null => P\_x1.AddElement(P\_a1)

P\_a1 != null AND P\_x1.CanNotAddCars => OP1.SendOverNetwork(“ini”)

=> OP2.SendOverNetwork(“ini”)

=> OP3.SendOverNetwork(“ini”)

=> OP4.SendOverNetwork(“ini”)

=> ini.MakeNull(“”)

t1: r1r2r3r4.!=null => g1r2r3r4= r1r2r3r4

=> OP1.SendOverNetwork(“green”)

same for t4, t6, t8

t2: g1r2r3r4 != null AND in1!=null => g1r2r3r4=y1r2r3r4

=> OP1.SendOverNetwork(“yellow”)

=> t2.DynamicDelay(“Ten”)

g1r2r3r4!=null AND in1==null => g1r2r3r4=y1r2r3r4

=> OP1.SendOverNetwork(“yellow”)

=> t2.DynamicDelay(“Five”)

same t5, t7

t3: y1r2r3r4!=null => y1r2r3r4=r1g2r3r4

=> OP1.SendOverNetwork(“red”)

=> OP2.SendOverNetwork(“green”)

t9: r1r2r3y4!=null => r1r2r3y4 = r1r2r3r4

**Controllor 2**

**Place Types**

Five, Ten– Data Integer

Ini, red, green, yellow, r1r2r3r4, g1r2r3r4, y1r2r3r4, r1g2r3r4, r1r2g3r4, r1r2y3r4, r1r2r3g4, r1r2r3y4, in5,in6,in7, in8– Data String

OP5, OP6, OP7, OP8, – Data Transfer

**Grd&map**

iniT : ini != null

=> OP5.SendOverNetwork(“ini”), OP6.SendOverNetwork(“ini”), OP7.SendOverNetwork(“ini”), OP8.SendOverNetwork(“ini”),ini.MakeNull(“ ”)

t1 : r1r2r3r4 != null

=> r1r2r3r4 = g1r2r3r4, OP5.SendOverNetwork(“green”)

t2: g1r2r3r4.NotNull AND in5.NotNull

=> OP5.SendOverNetwork(“yellow”), g1r2r3r4=y1r2r3r4, t2.DynamicDelay(“Ten”)

g1r2r3r4.NotNull AND in5.isNull

=> OP5.SendOverNetwork(“yellow”), g1r2r3r4=y1r2r3r4, t2.DynamicDelay(“Five”)

\*same for t4, t6,t8

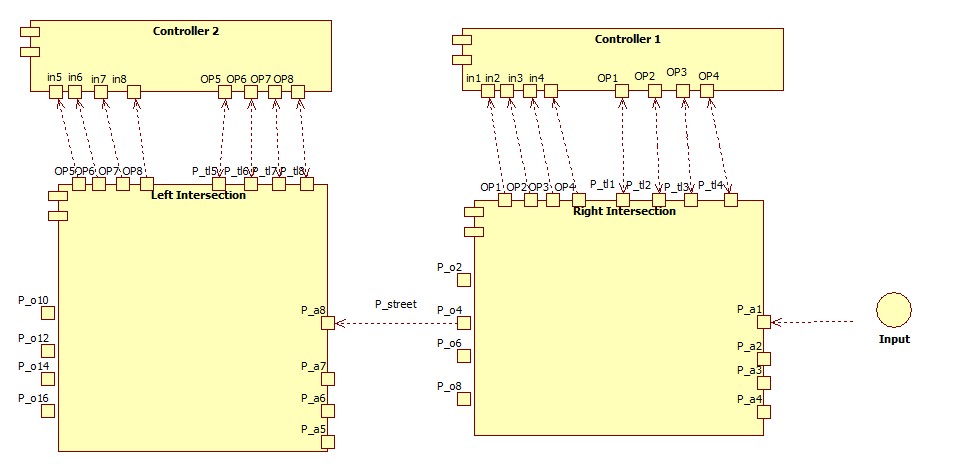
t3: y1r2r3r4 != null

=> y1r2r3r4=r1g2r3r4 , OP5.SendOverNetwork(“red”), OP6.SendOverNetwork(“green”),

\*same for t5,t7

t9: r1r2r3y4 != null => r1r2r3y4=r1r2r3r4

1. The component diagram for the entire system (depending on your implementation, each OETPN is considered a component) and show the names of the input and output channels.



**Implementation:**

1. Link for the repository:

<https://github.com/couman05/DCS-Intersection-Project>

**Testing:**

1. Send a car from the 1st intersection, that should go through the middle street and exit from one of the exit lanes from the 2nd intersection. Attach screen shots showing how the car moves and at the end of the test, pause the intersection OETPN and click on the save log button, save it as test1\_intersection 1.txt and test1\_intersection 2.txt if you have implemented them in two separate OETPNs. Then add the text file/s to the repository.
2. Traffic jam: for each intersection, create a traffic jam case by sending the maximum number of cars to the input lane of the intersection, start the controller, then send the last car. The controller should receive a signal from the plant (intersection) and the transition that is responsible for sending a yellow light to that lane where you input the cars to, should have changed the delay to 10 sec. Let the controller OETPN run until it reaches the same transition (2 loops) to show that the delay is changed back to 5 sec. pause the controller OETPN and click on the save log button, save it as test2.txt and add the text file to the repository.