EMBSYS 110 Assignment 3

Traffic Light

1 Goals

To become proficient in using statecharts to describe system behaviors and to gain experience in implementing orthogonal regions in QP.

2 Readings

a) **Systems of Systems Modeled by a Hierarchical Part-Whole State-Based Formalism**. Luca Pazzi. June 2013.

(File: System of Systems.pdf)

This paper discusses how to partition a system into layers of HSMs. At each layer, an HSM is a ***part*** working for the HSM at the layer above it, while at the same time acts as a ***whole*** representing its constituent HSMs at the layer below it.

Fig. 3 shows the traffic light system similar to that we use in this assignment. The controller (bottom part – Note it's upside down) is a part to the bigger system while it is a whole representing the two lights it controls/synchronizes.

Fig. 2 shows the counter-example without using this hierarchical control pattern in which two traffic lights interacting directly to each other without an explicit controller.

b) **From Protocol Specification to Statechart to Implementation**. Kiri L. Wagstaff, Ken Peters, and Lucas Scharenbroich. Jet Propulsion Laboratory. October 2008.

(File: From Protocol Specification to Statechart to Implementation.pdf)

Like (a) above this paper uses a traffic light system to illustrate statecharts and orthogonal regions. The Traffic component in our assignment project is based on Figure 2 of this paper.

3 Setup

You can skip these setup steps if you have already completed them.

1. Download the project compressed file (platform-stm32f401-nucleo_assignment_3_traffic.tgz) from:

   https://canvas.uw.edu/courses/1188665/files/folder/Assignments/Assignment%203%20Traffic

Place the downloaded tgz file in **~/Projects/stm32**.

2. Make sure you backup your existing project folder, e.g.

   mv platform-stm32f401-nucleo platform-stm32f401-nucleo.bak

3. Decompress the tgz file with:

   tar xvzf platform-stm32f401-nucleo_assignment_3_traffic.tgz

   The project folder will be expanded to **~/Projects/stm32/platform-stm32f401-nucleo**.

4. Launch Eclipse. Hit F5 to refresh the project content.

   Or you can right-click on the project in Project Explorer and then click "Refresh".

5. Clean and rebuild the project. Download it to the board and make sure it runs.

   In minicom, you should see a new command named "traffic". Try out the command "traffic n" or "traffic e". You should see log messages printed out on the console.

6. If you have not installed UMLet, you will need to do so for this assignment. You can download the latest version from:

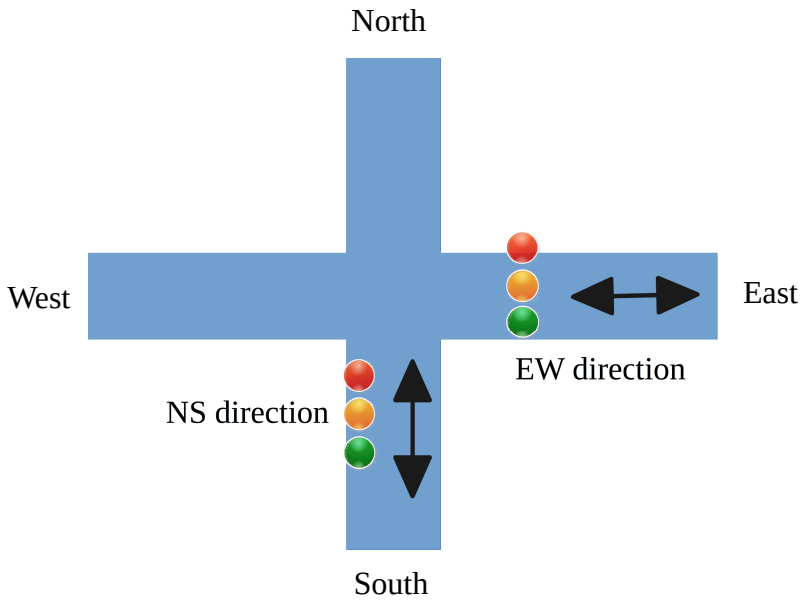
   http://umlet.com/changes.htm

   The stand-alone version is more stable. It runs on Java, so it should work in your VM which has Java installed.

4 Tasks

The design used in this assignment is for a traffic light system at an intersection between North-South (NS) and East-West (EW) streets.

There are two sets of traffic lights, one along the NS direction and the other along the EW direction. They are illustrated in the diagram below. Note that there are two opposite facing traffic lights in each set, which show the same pattern (both RED or both GREEN, etc). For brevity, only one in each set (south facing and east facing) is shown below. Also for simplicity there are no left-turn signals.



1. In the assignment project, the traffic light system is located in the folder src/Traffic. It contains the Traffic active object and the Lamp orthogonal regions (under src\Traffic\Lamp").

   Note – The event TRAFFIC_CAR_NS_REQ is a request from another system component (e.g. sensor) to request passage along the NS direction. It is supposedly generated each time when a car is detected to arrive in front of the north or south facing traffic light. The similar concept applies to the EW direction.

   For simulation, the console command '**traffic n**' or '**traffic s**' generates a TRAFFIC_CAR_NS_REQ event, and '**traffic e**' or '**traffic w**' generates a TRAFFIC_CAR_EW_REQ event.

2. The reference design is described in the statechart named Traffic.uxf. Your are tasked to improve it according to the following new requirements:

   (a) Imposes a minimum duration between each change of direction to avoid the "go" direction from toggling too fast. Note that the NS direction is hypothetically busier and its minimum "go" duration (20s) is intentionally longer than that of the EW direction (10s).

   (b) Since the NS direction is the main route, we would like to have "go" direction automatically return to the NS direction when *no cars have been detected along the EW direction for 15 seconds*. Note: You can use the event TRAFFIC_CAR_EW_REQ to check if a car has passed through the intersection along the EW direction.

   (c) Do you see another problem with the current design? What if a TRAFFIC_CAR_EW_REQ event arrives in the EWSlow state? That is a EW-going car is detected while the yellow

light is shown along the EW direction. (Assume the driver behaves well and stops in front of the light rather than rushing through it.) Propose a solution to fix this issue to avoid the car from being stuck in front of the light forever.

3. Show your design changes in the Traffic statechart. Update the Traffic.uxf file with UMLet and export the finished design to a pdf file for submission.
4. (Optional) Implement your design changes in the source code.
5. (Optional) Test your code with the console command "Traffic ..." and ensure the expected log messages are observed.

5 Submission

The due date is 5/14 Monday 11:59pm. Please submit:

1. The pdf file of the modified statechart design (exported from UMLet).
2. (Optional) A zip file containing the source code in **src/Traffic**. If you have modified other folders you can include them as well.
3. (Optional) A log file capturing the output of the UART console. You can turn on capturing in minicom by:

   sudo minicom ***-C log.txt***

   Your log should demonstrate the two new requirements are fulfilled..
4. Upload to the usual Canvas assignment upload location.