

Department of Computer Science

Real-Time Systems

CS 4514 (Section 01)

Final Project

Ross Tebbetts, rtebbett@students.kennesaw.edu

Mitchell Arnold

(Tebbettsross@yahoo.com)

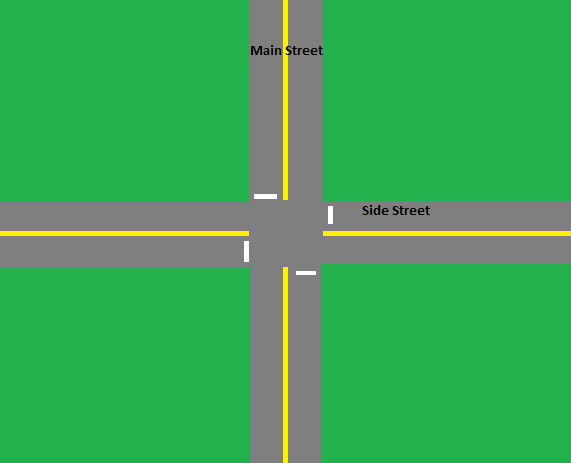
September 15th 2015

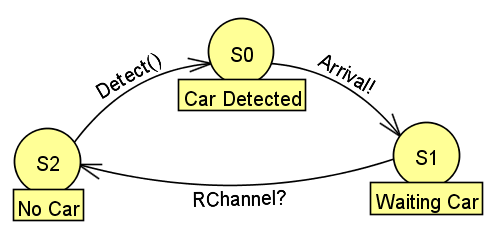
Traffic Light Simulation

The purpose of this project is to design and simulate a four way intersection and the traffic lights that direct drivers. A traffic light is an example of a hard real time system as missed deadlights can put drivers in danger so a simulation is necessary to test this system. The simulation was designed using the OOSimL simulation language and the resulting java code after compilation to design a simple GUI for the system.

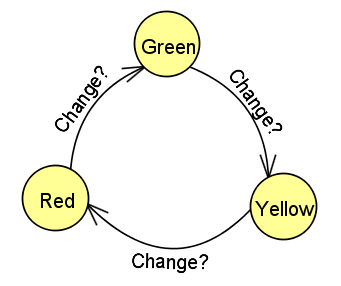
To explain how the system was designed first we must explain how the traffic light should work. The layout is a four way intersection consisting of four lights directing traffic for two roads Side Street and Main Street. For simplicities sake we did not include turn signals for drivers turning left or right. This choice was made because this system is a proof of concept for a traffic light and is intended to be expanded at a later date. The System consists of four different types of state machines. The controller handles the organization of changing the light and determining which road is moving by communicating with the other individual state machines. The monitor communicates with both the controller and the sensor through channels to detect incoming cars on one of the two streets. These last two state machines exist for each road in the simulation meaning one for Main Street and one for Side Street. The sensor handles detection of incoming vehicles waiting for the light to change and communicates with the Monitor via channel. The last state machine handles the color of the Light for its specific street. The reason for having only one sensor and light state machine for each street is because we have not yet implemented the ability for cars to turn left at the intersection. This means that when a car is detected on either side of a street it is safe for the light to change green for both directions of traffic on that road allowing one of each state machine to handle the incoming traffic.

The simulation was originally designed and written in OOSimL. Although the program functions correctly in purely OOSimL, it was necessary to use the compiled Java code to implement the GUI. For Demonstration purposes the GUI allows the user to modify certain aspects of the simulation to change the behavior during runtime. In the programs Current state these modifiers consist of simulation period, trace file name, stat file name, average interval of arrival for a car, deadline for communication between the light and controller, and variables handling detection of the car by the sensor. The GUI has text boxes to display the resulting trace and stat files for the Simulation after it has completed. There is much to add to the simulation but most of it was beyond our frame of time.

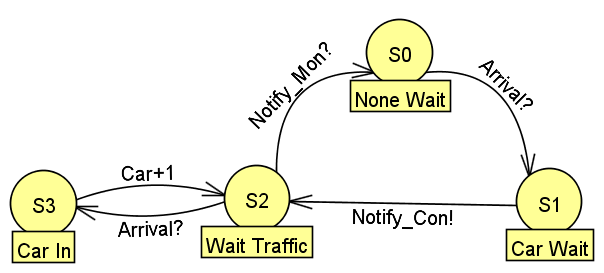




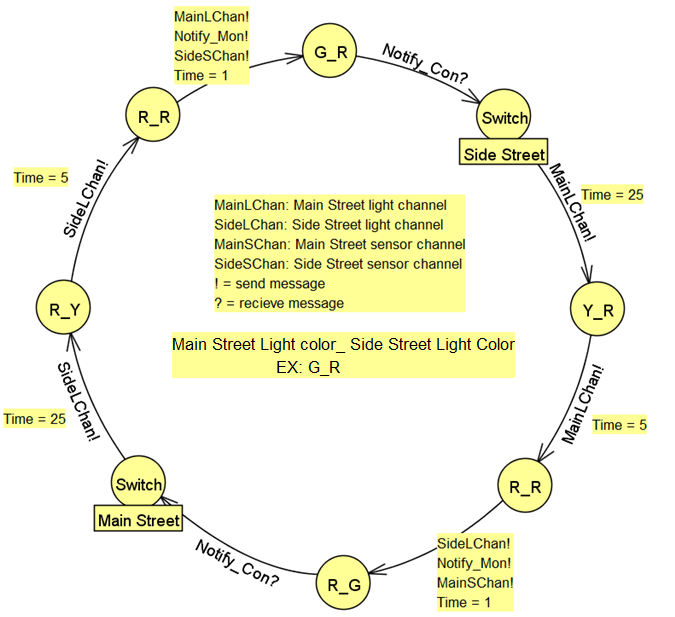
The Sensor State machine has three states Car Detected, Waiting Car, and No Car. It should be noted that there are two of these sensor Machines in the current simulation; one for each street. When one sensor state is active the sensor for the opposite road is not as traffic is currently flowing on that road and cars do not need to be detected. The sensor starts in the No Car state and actively listens for the detection of a car using the Detect( ) method. Once a Car is detected it switches to the Car Detected state and sends a message to the Monitor through the arrival channel telling it that a car has arrived. once a car has notified the monitor the sensor switches to the Waiting Car state where the sensor sleeps until notified by controller through RChannel to start detecting again. Once it is time to switch the light back to Red from Green, controller will wake up the waiting sensor to being detecting cars again.



The Light State machine is very simple. There is a light for each street Side and Main. Each light state machine determines the color of both traffic lights on that street allowing both directions to go. In all states the light is actively listening through the Change channel to be notified when the light needs to change. The states can only move in one direction meaning the light must be yellow after it is green and must be red after it is yellow and so on. The light only communicates with the controller.



The monitor starts in the None Wait state telling the monitor that there are currently no cars waiting. There is only one monitor in the simulation and it alternates monitoring the sensor of the road that is currently red. The monitor is aware of which street's sensor it is listening to and switches accordingly when woken up by the controller. While in the None Wait state the monitor is awaiting a message from the sensor saying that a Car has arrived and is currently waiting. Once this signal is received the state changes to Car Wait and it notifies the controller through the Notify\_Con channel and switches to the Wait Traffic State. The next state and function was part of the early development process and this Car In state is no longer necessary. This being said the monitor listens to the Sensor again and makes sure the sensor has detected a waiting vehicle and increments the Car variable by +1. After the controller has been notified the monitor sleeps until it receives a signal from the Controller through the Notify\_Mon channel to begin waiting for the sensors signal again.



The Controller state machine is the most detailed of the system because it handles the control of all sensors, lights, and the monitor. The state names in the diagram above represent the color of the lights for both of the streets. The left letter is the Main Street light state while the right is for Side Street. On start the simulation has Main Street Green and Side Street Red as shown by the state at the top of the diagram. In this state the controller is Listening to the Notify\_Con channel awaiting a signal from the monitor that a car has arrived on the Street that currently has a red light. After a signal is received the controller moves to the switch state where it waits for a period of 25 time units then notifies the Main street light through the MainLChan advancing its state to yellow. The wait is used in order to guarantee that once a light has turned green it will stay green for a minimum of 25 time units before changing even if the sensor instantly detects a car on the opposite road. After this the controller is now Y\_R where it waits for 5 time units then changes Main Street to red. At this point both lights are red and will be for at least one time unit. Next the controller notifies the three channels for Side street's light, the Main street sensor, and the monitor. This changes the Side street light to Green, wakes up the Main street sensor, and wakes up the monitor. After this The process repeats for the now changed lights. The controller's states represent the current state of both street's lights which guarantees that no light will ever be green at the same time.

Although our simulation functions above the simulations most basic form, it is far from polished. The simulation itself has room for improvement by developing lights for individual turn signals instead of only allowing traffic to flow straight through the light. The GUI deserves to be much more detailed than it currently is. We have plans to add a visual representation of the intersection in action through graphics displayed in the program. Lastly there is one true flaw of this program. The simulation runs and completes just fine; however, it does not allow the simulation to restart without closing the program and starting another instance. Through testing and debugging we were never able to fix this problem. We suspect a few things to be the cause of this obstacle. It is possible that the threads are not being reinitialized when a new instance of the Car object is created. The Car class acts as the main where all sensor, controller, monitor, and light objects are initialized. it is also possible that simply the Car class if failing to reinitialize when it is told to. The code to fix this problem is in the source code but its activation was commented out.

While writing this simulation an incredible amount was learned. Using OOSimL to write this simulation was miles easier than writing the whole program from scratch in java. After Reading through the Train Gate Simulation for guidance we decided to use the channel and message class that the simulation used and repurposed them for communication in our traffic light simulation. The Traffic light simulation compiles and functions in OOSimL but the fact that OOSimL compiles into java code allowed us to write a GUI using Oracle's Scene Builder. Although the GUI may be simple it allows a user friendly way to change simulation Modifiers without directly changing the source code and shows the results of the simulation. These features helped greatly with debugging as well as making the simulation easier to use. We learned a great deal about developing GUI's for a program as well as the conveniences of using a simulation language for design and testing of Real Time Systems.

Framework:

Oracle Scene Builder for GUI

OOSimL for overall Simulation design

Java for Developing GUI with Scene Builder

Credits

Dr. Jose Garrido: for Design of OOSimL, OOSimL compiler, Message and Channel class from the Train Gate Simulation.