14ELB014 - Software Engineering

UML Coursework Task

By James Moore (A910831) and

Devon Kerai (B128203)

Contents

System Overview

This report details a UML model for a three directional traffic junction containing a dual carriageway, filter lanes and pedestrian crossings; Figure 1 shows this in as a basic visual diagram.

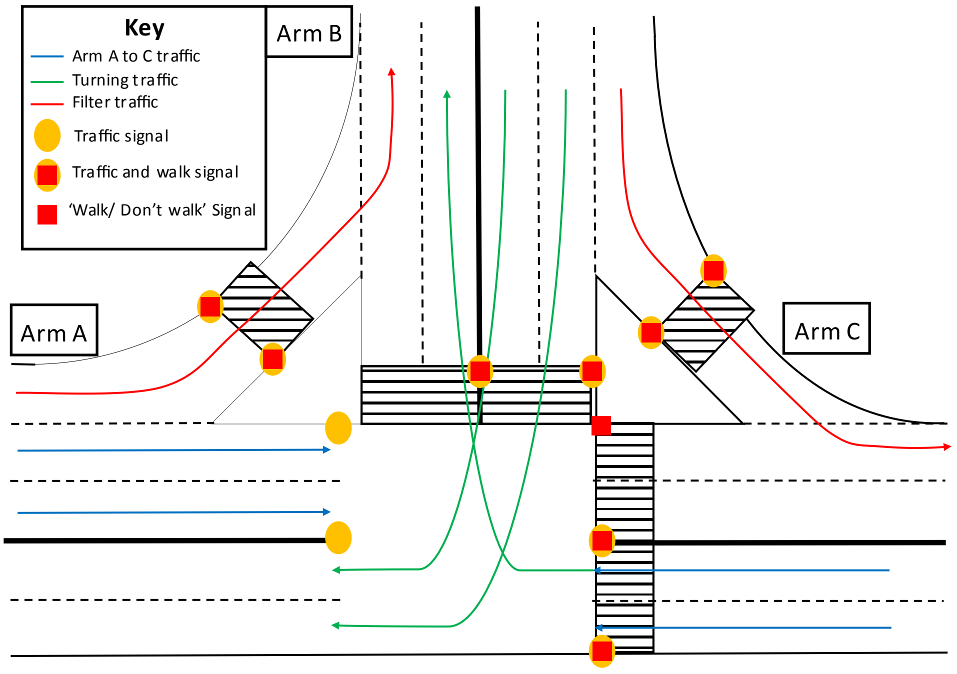


Figure 1 - Basic layout of junction

Two different types of traffic lights are shown in the diagram. A simple traffic signal, shown as a single yellow circle, a single walk/don't walk signal, shown as a red square and a traffic light signalcombined with a walk/ don't walk signal shown by a red square inside a yellow circle.

Arm A contains two types of lanes, a filter lane into Arm B and a dual carriageway into Arm C. The filter lane contains a traffic and walk signals whereas the dual carriageway contains a traffic signal only. Cars in the filter lane are assumed to merge independentlyinto the flow of traffic of Arm B.

Arm B also contains a filter lane into Arm C and a dual carriageway turning right into Arm A. Both lanes consist of traffic and walk signals. Cars in the filter lane are assumed to merge into the flow of traffic independently of Arm A.

Arm C contains a dual carriageway which continuesstraight into Arm A and a right turn lane into Arm B.The dual carriageway containstrafficand walk signal. Cars in the right turn lane are assumed to cross the junction independently.

The junction has been modelled using: A Use Case diagram, a Class diagram, Sequence diagrams, Object diagrams and a State Machine diagram.

Functionality

The junction consists of three main sections labelled:Arm A, B and Crespectively. The junction is able to operate under two different conditions: busy periods and quietperiods. Under normal operation (busy), traffic sensors in each lane allow the system to monitor how busy each junction is; the junction is determined as busy if carsare present across multiple lanes. The system will cycle the traffic lights for thirty seconds ongreen; as shown in Figure 1. Cars and pedestrians are assumed to move together.If a pedestrian wants to cross the movement of traffic, the junction is stopped and all pedestrians are able to cross.

During quiet periods, all traffic lights are set to red with all pedestrian crossings set to green. When a car is detected, the appropriate lights turn green allowing the car to pass before returning to red. While the traffic lights are set to red, all pedestrian walk signals are set to green.

Use Case Diagram

A Use Case diagram is used to represent the design of a system at a high level and how the users interact with it. The actors in this system are represented by a car and a pedestrian.

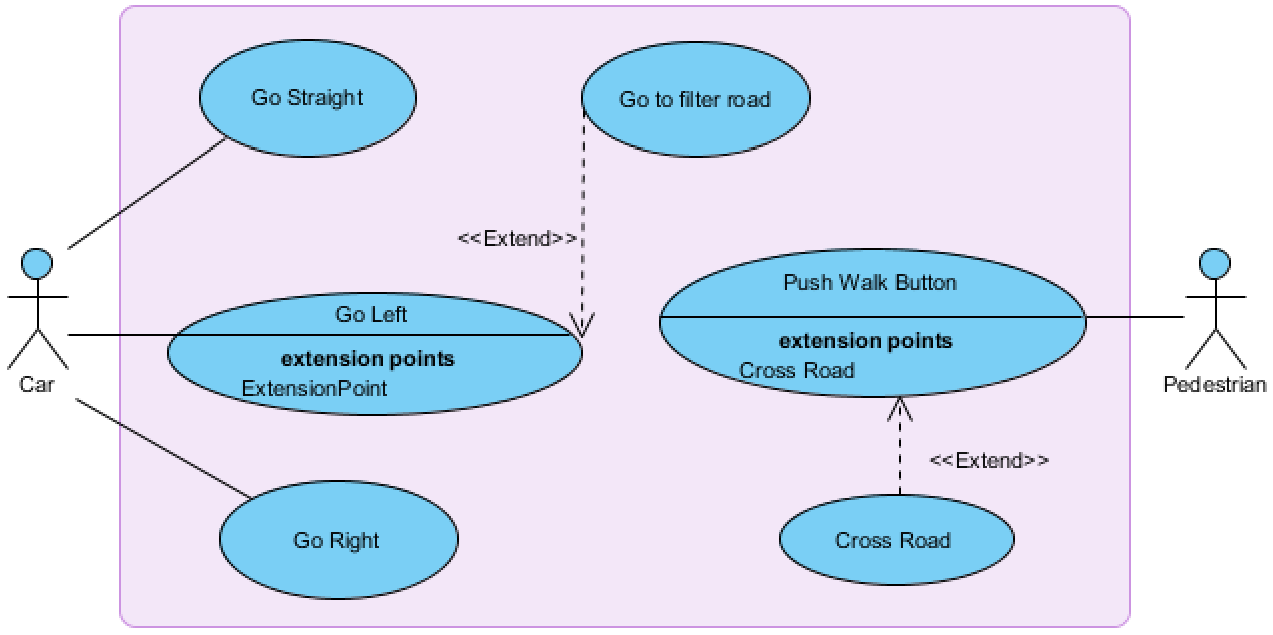


Figure 2 - Use Case Diagram

Figure 2 shows a car being able to either: Turn straight, turn right or turn left. If the car is turning left, it is classified as a sub scenario and therefore an <<Extend>> interaction is used to illustrate the use of a filter lane.

In order to cross thejunction, the pedestrian pushes the 'walk' button as shown in Figure 2. An <<Extend>> interaction is again used to show the relationship of crossing the road.

Class Diagram

Figure 3demonstrates the relationships amongst classes in a system, showing their operations and attributes. A number of different connection types were used within this system's class diagram. These are: composition, association and inheritance.

Composition shows that the associated items are dependent on the parent item. For example, the traffic light column consists of traffic lights, pedestrian lights and a pedestrian button meaning if the column is destroyed the associated items are also removed.

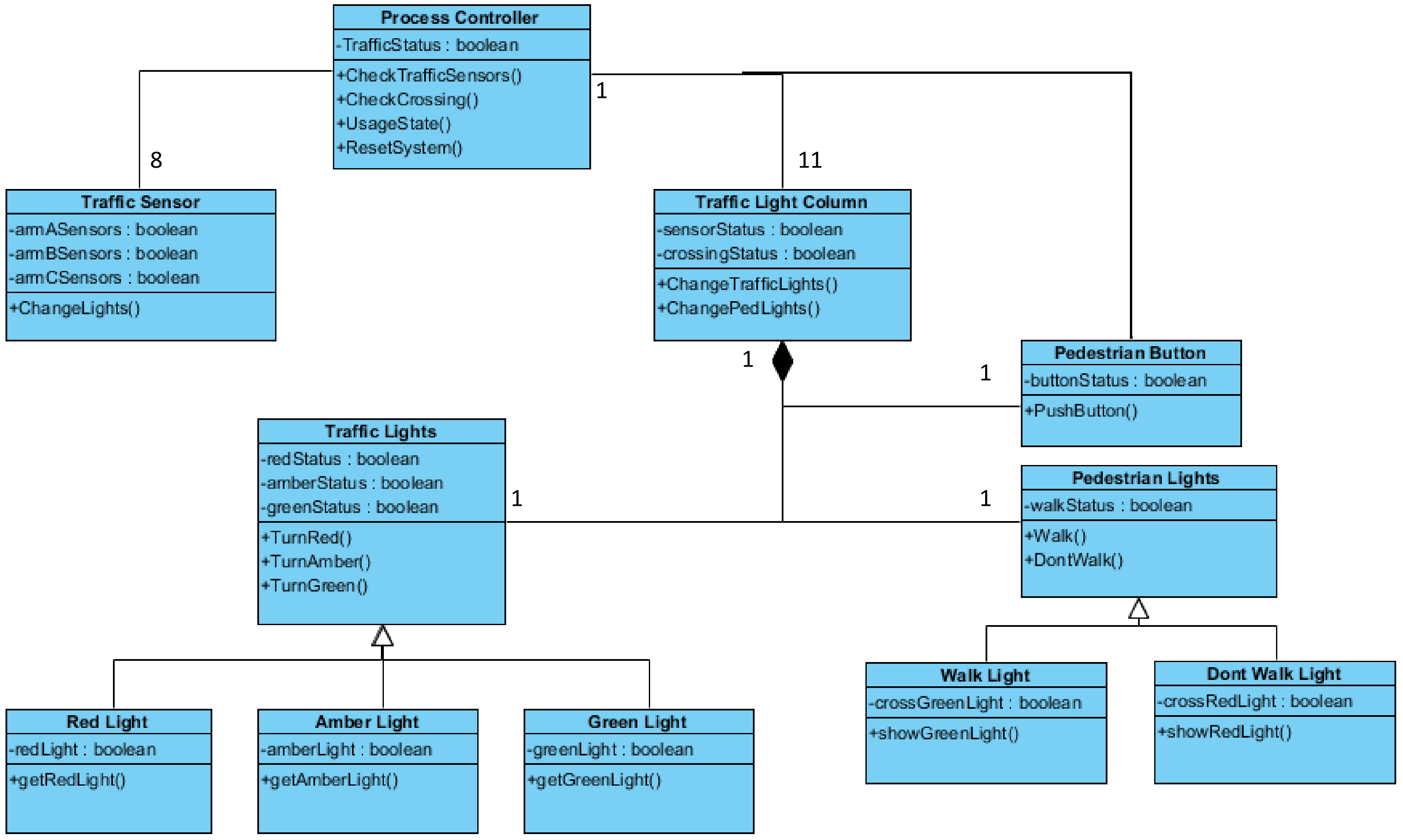


Figure 3 - Class Diagram

Association is the connection between two classes, with the number representing the number of instances of the class. One process controller corresponds to eight traffic sensors and eleven traffic light columns.

Inheritance is when a class contains a collection of other classes. A *Traffic Light* contains three different lights (Red, Amber and Green).

Process controller: Controls the entire system and contains four functions:*CheckTrafficSensors, CheckCrossing, UsageState*and *ResetSystem.* The *CheckTrafficSensors*function checks to see if any of the traffic sensors have been triggered throughout the system while the *CheckCrossing* function verifies if a pedestrian button has been pressed.*UsageState* returns a Boolean value indicating what statethe system is in: Busy or quiet.*ResetSystem* resets the entire traffic junction to the quiet state.

Traffic Sensor: Retrieves the values of the Arm A, B and C sensors and returns the state to the *Process Controller class.*

Traffic Light Column: Contains two functions which call other classes to change the traffic lights and pedestrian lights respectively.

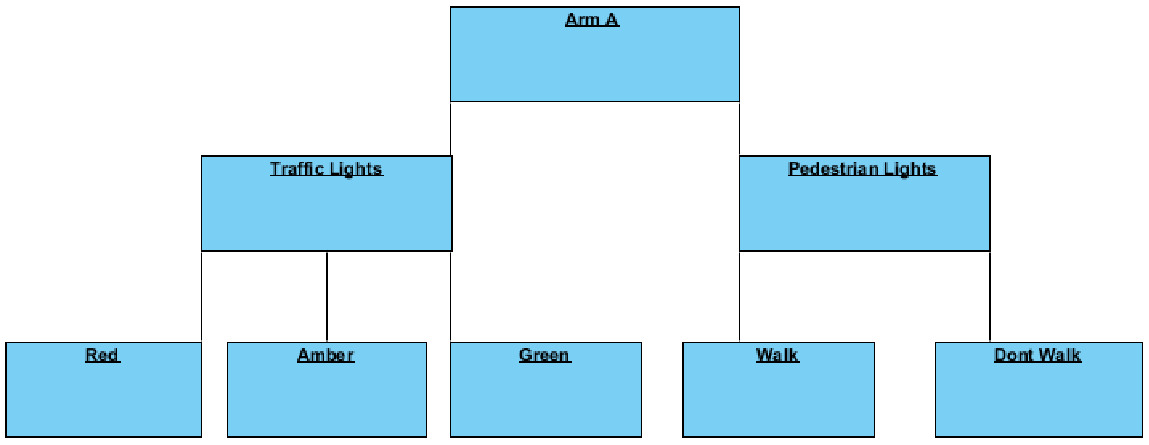
Pedestrian Button: Changes the state of the pedestrian button when pressed by a pedestrian.

Pedestrian Lights: A parent class made up of two sub-classes which changes the status of the pedestrian lights.

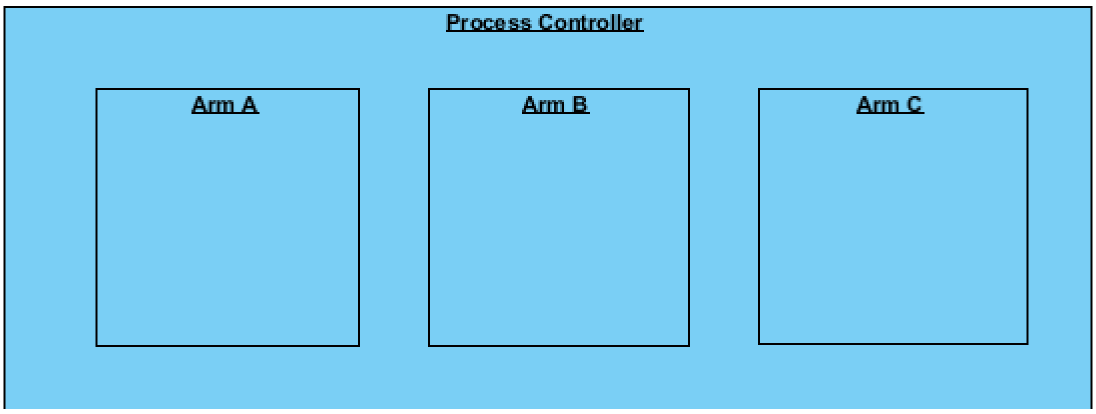
Traffic Lights: Parent class that inherits three Light sub-classes (Red Light, Amber Light and Green Light) that display the appropriate light on the traffic light column.

Object Diagram

Object diagrams focus on the links between instances within a system. Figure 4 (a) shows the static structure of an individual Arm within the complete system shown in Figure 4(b).



(a)



(b)

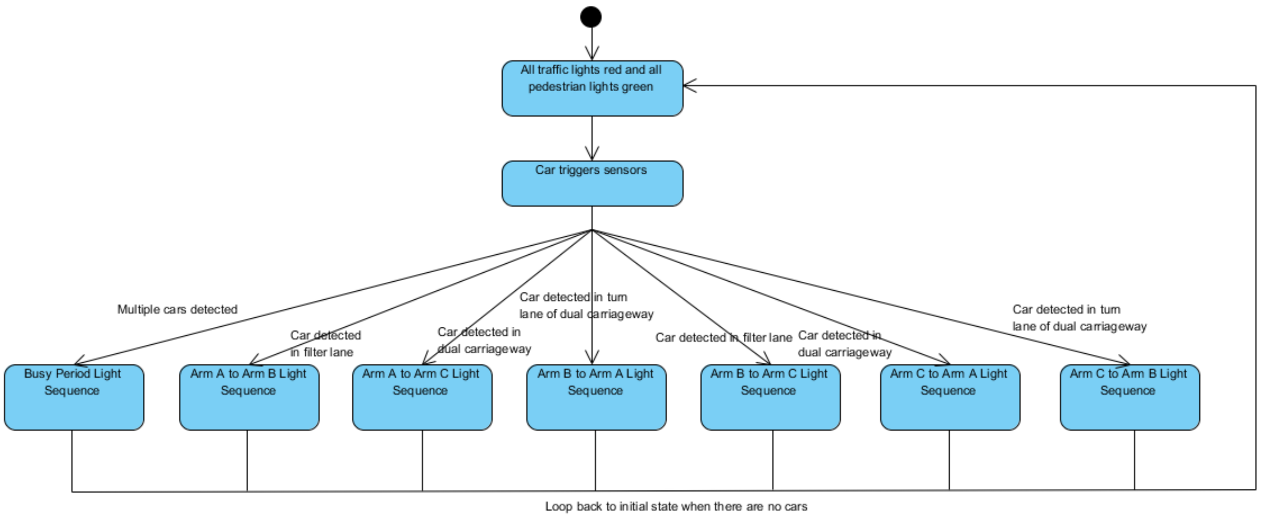
Figure 4 - Object Diagrams

State Machine Diagram

A State Machine diagram shows the different states within the system and how it transitions from one state to another. Figure 5 shows the State Machine diagram.

The first state is an initialisation stage and is the state used during quiet periods of traffic. During this state, all traffic lights are set to red and all the pedestrian crossings are set to green and the system continues to monitor sensors for traffic. If a car is detected by the sensors, the appropriate light sequence is triggered. Once the car has crossed the junction and no further cars are detected, the system returns to the initialisation state.

If multiple cars are detected, the system changes to the *Busy PeriodLight Sequence* state, in which the traffic lights are cycled, shown in the following sequence diagrams.It then returns to the initialisation state when no further cars are detected. The state diagram does not terminate, but instead loops indefinitely.



All traffic lights red and all pedestrian lights green
Car triggers sensors
Multiple cars detected
Car detected in filter lane
Car detected in dual carriageway
Car detected in turn lane of dual carriageway
Car detected in filter lane
Car detected in dual carriageway
Car detected in turn lane of dual carriageway
Busy Period Light Sequence
Arm A to Arm B Light Sequence
Arm A to Arm C Light Sequence
Arm B to Arm A Light Sequence
Arm B to Arm C Light Sequence
Arm C to Arm A Light Sequence
Arm C to Arm B Light Sequence
Loop back to initial state when there are no cars

Figure 5 - State Diagram

Sequence Diagram

Sequence diagrams are used to illustrate the flow of the system, showing the processes' order and how they interact with one another. Although there are six distinct ways of crossing the junction, to minimise repetition, only five diagrams have been used: Arm A to Arm C (straight across), Arm A to Arm B (filter lane), Arm B to Arm A(turning right) and busy period sequences for both cars and pedestrians.

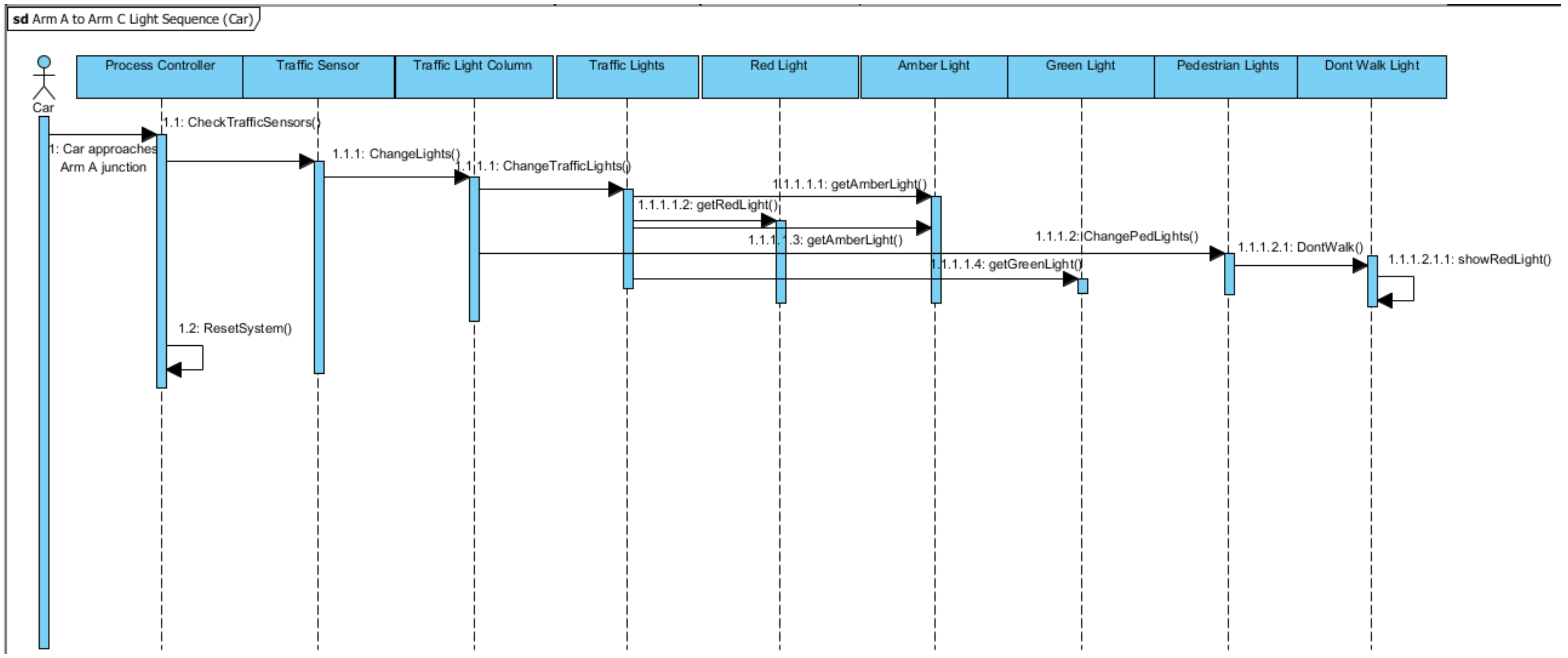

sd Arm A to Arm C Light Sequence (Car)
Car
Process Controller
Traffic Sensor
Traffic Light Column
Traffic Lights
Red Light
Amber Light
Green Light
Pedestrian Lights
Dont Walk Light
1: Car approaches Arm A junction
1.1: CheckTrafficSensors()
1.1.1: ChangeLights()
1.1.1.1: ChangeTrafficLights()
1.1.1.1.1: getAmberLight()
1.1.1.1.2: getRedLight()
1.1.1.1.3: getAmberLight()
1.1.1.1.4: getGreenLight()
1.1.1.2: ChangePedLights()
1.1.1.2.1: DontWalk()
1.1.1.2.1.1: showRedLight()
1.2: ResetSystem()

Figure 6 - Arm A to Arm C Sequence Diagram

Figure 6 shows the sequence for cars traveling from Arm A to Arm C during the quiet period. As a car approaches the Arm A junction, it is detected by the sensor. The process controller then changes the lights accordingly. The red lights are changed through the sequence: red - red and amber - green, while all the pedestrian crossings are changed to red by calling *ChangePedLights*from the *Traffic Light Column* class. Once the car has passed, the system is reset to the initialisation state via *ResetSystem*.

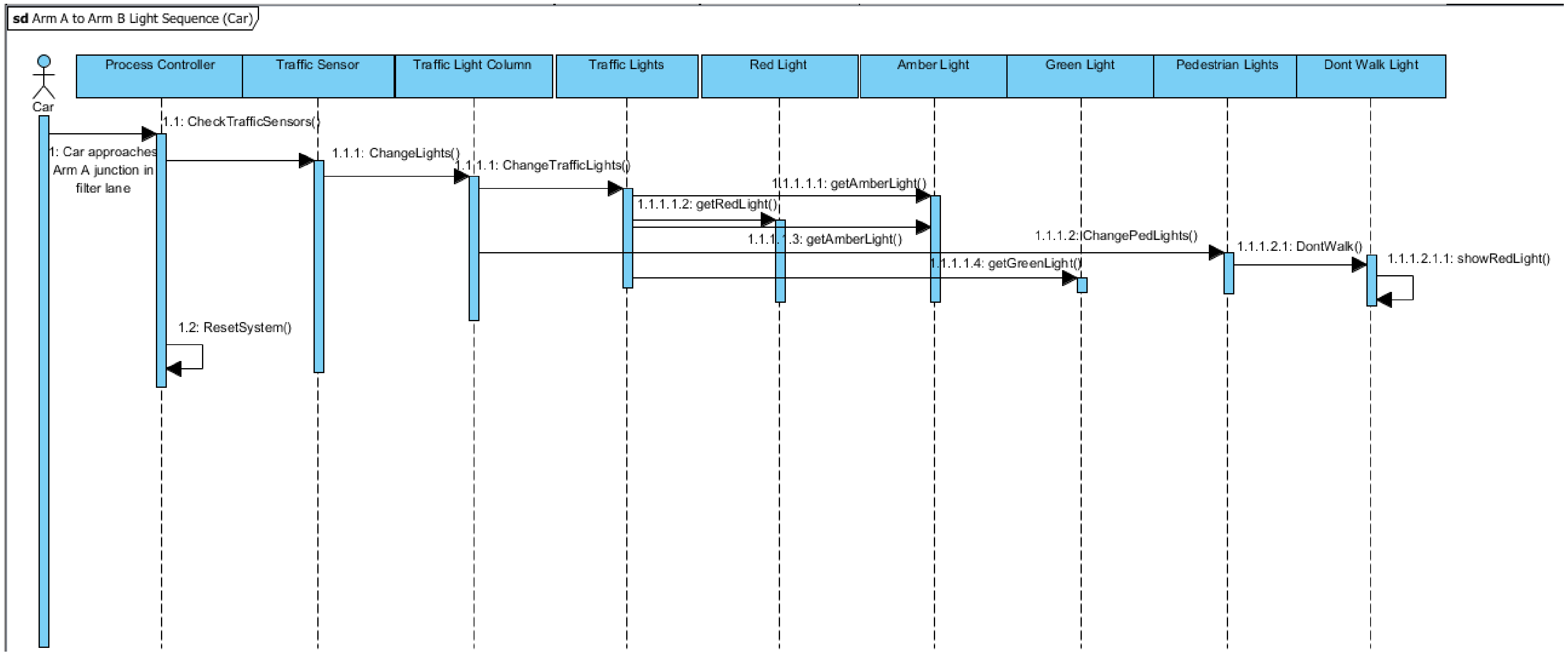

sd Arm A to Arm B Light Sequence (Car)
Car
Process Controller
Traffic Sensor
Traffic Light Column
Traffic Lights
Red Light
Amber Light
Green Light
Pedestrian Lights
Dont Walk Light
1: Car approaches Arm A junction in filter lane
1.1: CheckTrafficSensors()
1.1.1: ChangeLights()
1.1.1.1: ChangeTrafficLights()
1.1.1.1.1: getAmberLight()
1.1.1.1.2: getRedLight()
1.1.1.1.3: getAmberLight()
1.1.1.1.4: getGreenLight()
1.1.1.2: ChangePedLights()
1.1.1.2.1: DontWalk()
1.1.1.2.1.1: showRedLight()
1.2: ResetSystem()

Figure 7 - Arm A to Arm BSequence Diagram

Figure 7shows the sequence for cars traveling from Arm A to Arm B during the quiet period. As a car approaches the filter lane in the Arm A junction, it is detected by the sensor. The process controller then changes the lights accordingly. The red lights are changed through the sequence: red - red and amber - green, while all the pedestrian crossings are changed to red by calling *ChangePedLights*from the *Traffic Light Column* class. Once the car has passed the system is reset to the initialisation state via *ResetSystem*.

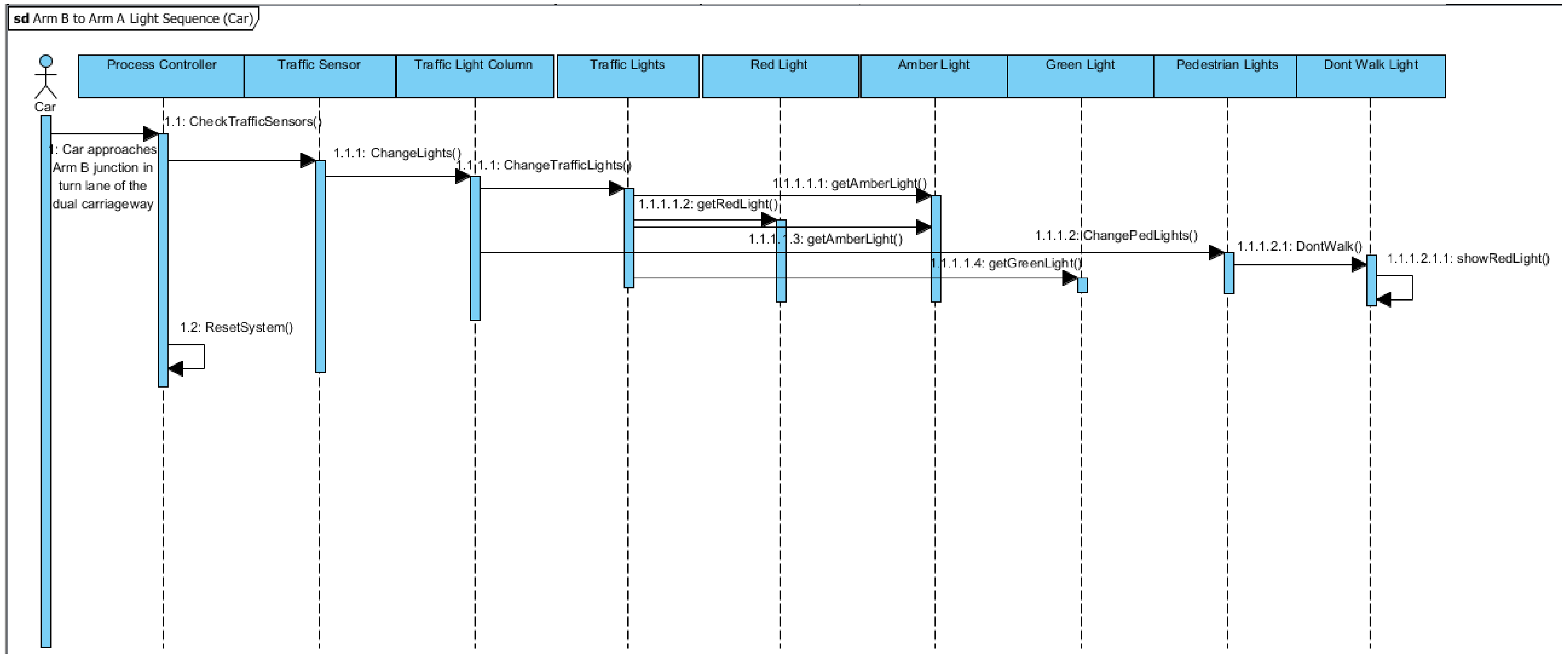

sd Arm B to Arm A Light Sequence (Car)
Car
Process Controller
Traffic Sensor
Traffic Light Column
Traffic Lights
Red Light
Amber Light
Green Light
Pedestrian Lights
Dont Walk Light
1: Car approaches Arm B junction in turn lane of the dual carriageway
1.1: CheckTrafficSensors()
1.1.1: ChangeLights()
1.1.1.1: ChangeTrafficLights()
1.1.1.1.1: getAmberLight()
1.1.1.1.2: getRedLight()
1.1.1.1.3: getAmberLight()
1.1.1.1.4: getGreenLight()
1.1.1.2: ChangePedLights()
1.1.1.2.1: DontWalk()
1.1.1.2.1.1: showRedLight()
1.2: ResetSystem()

Figure 8 - Arm B to Arm ASequence Diagram

Figure 7 shows the sequence for cars traveling from Arm B to Arm Aduring the quiet period. As a car approaches the right turn lane in the Arm B junction, it is detected by the sensor. The process controller then changes the lights accordingly. The red lights are changed through the sequence: red - red and amber - green, while all the pedestrian crossings are changed to red by calling *ChangePedLights*from the *Traffic Light Column* class*.* Once the car has passed the system is reset to the initialisation state via *ResetSystem*.

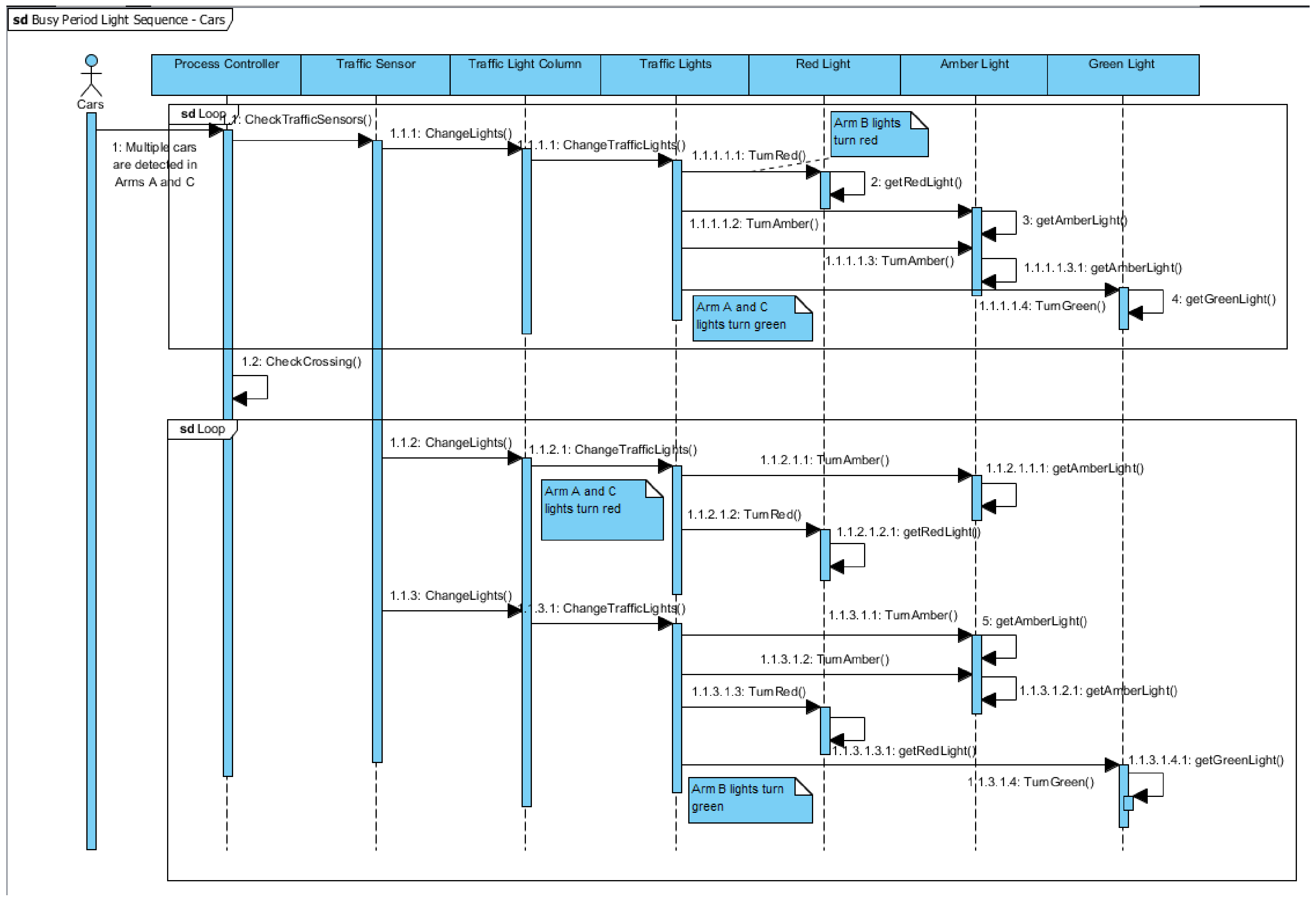


Figure 9 - Busy Period (Cars) Sequence Diagram

During busy periods, the traffic lights cycle as shown by the sequence diagram in Figure 9. Upon multiple cars being detected, the Arm B traffic lights are turned red whilst Arm A and Arm C are sequenced to green, occurring every thirty seconds, allowing traffic to flowconstantly.

Once the first loop has been executed and no pedestrian crossing requests have been called, a second thirty-second loop is ran. The Arm A and Arm C traffic lights are now sequenced to red, stopping traffic flowing, whilst the Arm B traffic lights are changed to green.

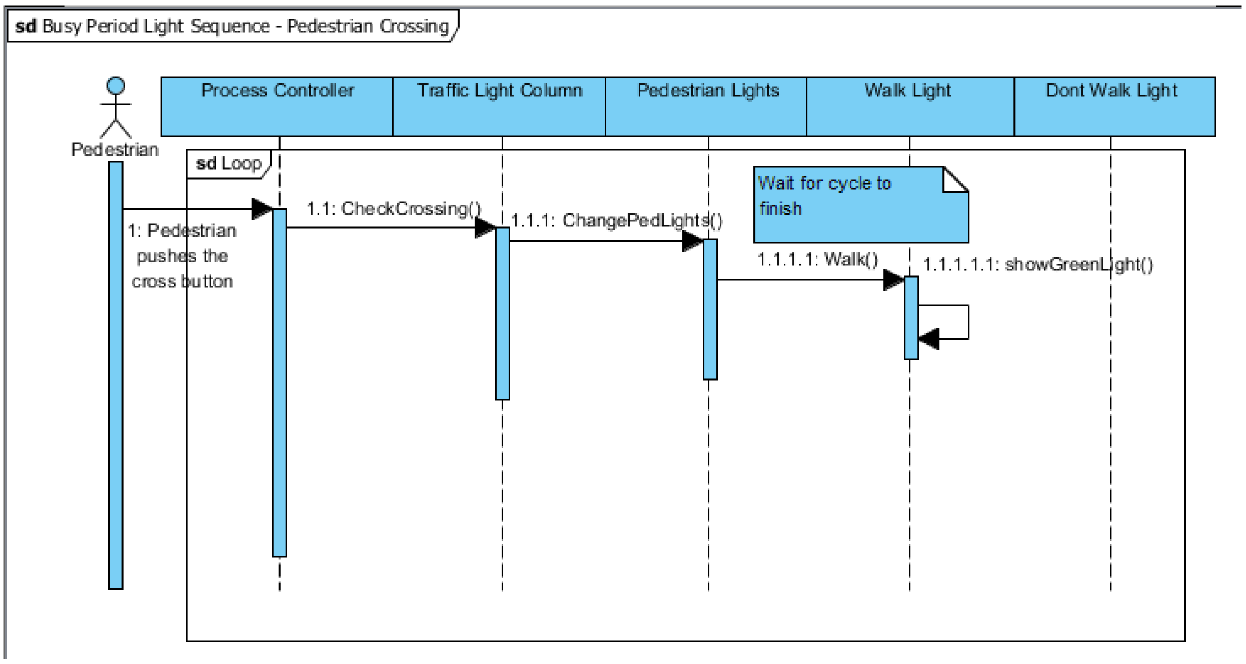


Figure 10 - Busy Period (Pedestrians) Sequence Diagram

If at any period during the busy light sequence shown in Figure 9 a pedestrian crossing request is made, the current light sequence is allowed to finish, therefore pedestrians are never waiting longer than thirty seconds. In between the light sequence changing, all traffic lights turn to red and the pedestrians are allowed to cross. This sequence is shown in Figure 10.

Conclusion

In conclusion the traffic light junction has been modelled using UML which includes traffic and pedestrian signals. A use case diagram, a class diagram, a state machine diagram, object diagrams and a series of sequence diagrams were used to show the behaviour of the system. The use case diagram shows a system overall from the users viewpoint while the class diagram shows the structure and components of the complete system. The state machine diagram shows the different states of the systems and how movement between states is undertaken. Sequence diagrams show the timing aspects of the complete system and are how individual events are called. These diagrams can be considered the closest to how the system's code would be implemented.