Traffic Light System

Real-time Software Engineering Lab 1

Eslam El-Sherbieny
eselsh@ttu.ee
IVSM 166778

Tallinn University of Technology 2017

Table of Contents

1. Vision

A software for traffic light system using model-based principles and workflows. A main controller acting as the program mainframe, responsible for controlling traffic flow and managing synchronization between vehicle and pedestrian traffic flows. Using timed constraints and user requests to change the traffic flow between them. Following the realistic traffic light system with pedestrian traffic user requests as in real life applications. Technologies used are: Eclipse UML modeling solution, Ecore, Java, etc… Junit and other methods for testing.

1.1. **System:** Traffic Light System
1.2. **Problem statement:** A system allowing pedestrians to safely cross a street
1.3. **User:** Pedestrians and Car drivers
1.4. **Behaviour:** Time and request based system, traffic lights changes either when time limit is reached or when a user request it
1.5. **Controller objects:** Main controller to drive the lights for cars (red, yellow, green) and pedestrians (red, green)
1.6. **Safety analysis:** Malfunction of the system can cause traffic accidents. Potentially death of several persons.
1.6.1. **Safety class:** Catastrophic

2. Requirements

2.1. HLR

2.1.1. Functional Requirements

2.1.1.1. The system shall allow pedestrians to safely cross a street.
2.1.1.2. The system's scope is a controller that drives the lights for cars (red, yellow, green) and pedestrians (red, green).
2.1.1.3. The system is equipped with two buttons for pedestrians, one on each side of the road.
2.1.1.4. The lights for cars and pedestrians must never be green at the same time
2.1.1.5. A timer sends a pulse to the controller every second.
2.1.1.6. Alternation of the traffic lights follows the standard light regime.
2.1.1.7. In the default state the traffic light is green for cars and red for pedestrians
2.1.1.8. Reception of a button press shall initiate the sequence for changing the lights to green for pedestrians and red for cars
2.1.1.9. Green light for pedestrians shall stay on for X seconds
2.1.1.10. Green light for cars shall stay on for at least Y seconds

2.1.2. Non Functional Requirements

2.1.2.1. The system will follow the time constraints precisely, to maintain synchronous changes between cars and pedestrians traffic
2.1.2.2. The system shall take traffic change requests in a timely manner, queuing the button_pressing requests and executing them after calculating the current traffic status
2.1.2.3. The system will guarantee that at no time, car and pedestrian traffic shall be green at the same time.
2.1.2.4. In case of glitches, the system shall check the current status and act accordingly, reaching the default state where cars are green and pedestrians are red
2.1.2.5. In failure, the system should be fixed in a practical time frame
2.1.2.6. The system must be online 24/7
2.1.2.7. The system should consist of a controller with the main functionality and logic, a timer, a light container, a traffic light entity responsible for changing the lights, another for pedestrians, and finally another two for the buttons A and B.

2.2. LLR

2.2.1. Controller

2.2.1.1. Description

The main program logic resides here, this class will handle the synchronization protocol between various elements of the program

2.2.1.2. Responsibilities

- 2.2.1.2.1. Synchronization between Pedestrians and Vehicle traffic
- 2.2.1.2.2. Time tracking of each traffic state interval
- 2.2.1.2.3. Switching states between Pedestrians and Vehicles
- 2.2.1.2.4. Switching Traffic lights states
- 2.2.1.2.5. Responding to signals from buttons
- 2.2.1.2.6. Calculating safety measures for both Pedestrian and Vehicle traffic to determine the correct time for traffic switching and interval of each state

2.2.1.3. Attributes

- 2.2.1.3.1. Time: Integer
  An integer counter that holds the current time value
- 2.2.1.3.2. pedlight: PedLight
  Object containing the pedestrian lights information and status
- 2.2.1.3.3. carlight: CarLight
  Object containing the vehicle lights information and status
- 2.2.1.3.4. car_traffic_active: Boolean
  Boolean variable to define current traffic status of vehicle
- 2.2.1.3.5. ped_traffic_active: Boolean
  Boolean variable to define the current traffic status of pedestrians
- 2.2.1.3.6. buttonA_pressed: Boolean
  Boolean variable to define if the button A is pressed or not
- 2.2.1.3.7. buttonB_pressed: Boolean
  Boolean variable to define if the button B is pressed or not
- 2.2.1.3.8. car_green_time: Integer
  Integer variable to count the time spent for active vehicle traffic, when limit is reached, it should fire the traffic call and to be reset afterwards and initiated again when vehicle traffic light becomes green

2.2.1.3.9. ped_green_time: Integer
Integer variable to count the time spent for active pedestrian traffic, when limit is reached, it should fire the traffic call and to be reset afterwards and initiated again when pedestrian traffic light becomes green, **Note:** pedestrian light can go green either through time, or button request

2.2.1.4. Operations

2.2.1.4.1. switch_lights_buttonA()
This function will call switch_ped_light function when called to immediately initiate a sequence to turn pedestrian light to green.

2.2.1.4.2. switch_lights_buttonB()
This function will call switch_ped_light function when called to immediately initiate a sequence to turn pedestrian light to green.

2.2.1.4.3. timer_reset()
Reset the time

2.2.1.4.4. init()
Initialize the default values (not used)

2.2.2. Timer

2.2.2.1. Description
A time object, initiated with program start and keep a timestamp count

2.2.2.2. Responsibilities

2.2.2.2.1. Keeping time count of the overall process

2.2.2.2.2. Sending pulse signal to the controller every second

2.2.2.3. Operations

2.2.2.3.1. tick()
Sends new time value on each call

2.2.2.3.2. timer_reset()
Reset the time

2.2.3. Enumerations

2.2.3.1. Lights

2.2.3.1.1. Description
Enumeration holding the three lights variables {Red, Yellow, Green}

2.2.3.1.2. Attributes

2.2.3.1.2.1. Red: enumeration literal
2.2.3.1.2.2. Yellow: enumeration literal
2.2.3.1.2.3. Green: enumeration literal

2.2.4. PedLight

2.2.4.1. Description
Object containing the pedestrian light info
2.2.4.2. Attributes
2.2.4.2.1. ped_current_light: Lights
Integer value containing the current status of the light, default is 0 for Red
2.2.4.3. Operations
2.2.4.3.1. switch_ped_light()
Start switching sequence from one state to the next, **Note:** if the status is green it will return to yellow then red - this will be done in sync with switch_car_light()

2.2.5. CarLight

2.2.5.1. Description
Object containing the vehicle light info
2.2.5.2. Attributes
2.2.5.2.1. car_current_light: Lights
Integer value containing the current status of the light, default is 2 for Green
2.2.5.3. Operations
2.2.5.3.1. switch_car_light()
Start switching sequence from one state to the next, **Note:** if the status is green it will return to yellow then red - this will be done in sync with switch_ped_light()

2.2.6. Buttons A & B

2.2.6.1. Description
Objects for button A and B that represents the buttons that would activate abnormal traffic switch from vehicle to pedestrians
2.2.6.2. Responsibilities
2.2.6.2.1. Receive button push from user to activate switch

2.2.6.3. Attributes

2.2.6.3.1. activate():
When button pressed this function call the traffic switch functions to change traffic on the respective side from vehicle to pedestrians

3. Test Case Descriptions

3.1. Case 1

3.1.1. Test case identification → TC-001
3.1.2. Test case revision history → V1.0 Initial version
3.1.3. ID of software under test → ControllerTest.java
3.1.4. Test description → Controller functionality test
3.1.5. Requirements tested → HLR-2.1.1.2, LLR-2.2.1
3.1.6. Test category → High and low level
3.1.7. Test inputs →
3.1.7.1. setCar_traffic_active(true);
3.1.7.2. setPed_traffic_active(false);
3.1.7.3. setButtonA_pressed(true);
3.1.7.4. setButtonB_pressed(true);
3.1.7.5. getCarlight().add(cl1);
3.1.7.6. getCarlight().add(cl2);
3.1.7.7. getPedlight().add(pl1);
3.1.7.8. getPedlight().add(pl2);
3.1.8. Test outputs → Traffic lights status change, Cars (from Green to Red), and Pedestrians (from Red to Green)
3.1.9. Test steps → Please refer to sequence diagram
3.1.10. Pass/fail → Passed

3.2. Case 2

3.2.1. Test case identification → TC-002
3.2.2. Test case revision history → V1.0 Initial version
3.2.3. ID of software under test → TimerTest.java
3.2.4. Test description → Timer functionality test
3.2.5. Requirements tested → HLR-2.1.1.5, HLR-2.1.2.1, LLR-2.2.6
3.2.6. Test category → High and low level
3.2.7. Test inputs →
    3.2.7.1. setFixture(SysteshFactory.eINSTANCE.createTimer());
    3.2.7.2. int x = getFixture().tick();
    3.2.7.3. int y = getFixture().tick();
    3.2.7.4. getFixture().tick();
    3.2.7.5. getFixture().timer_reset();
3.2.8. Test outputs → Time increments successfully, Time is reset successfully
3.2.9. Test steps → Please refer to sequence diagram
3.2.10. Pass/fail → Passed

3.3. Case 3

3.3.1. Test case identification → TC-003
3.3.2. Test case revision history → V1.0 Initial version
3.3.3. ID of software under test → ButtonATest.java
3.3.4. Test description → Button A functionality test
3.3.5. Requirements tested → HLR-2.1.1.3, HLR-2.1.1.8 , LLR-2.2.6
3.3.6. Test category → High and low level
3.3.7. Test inputs →
    3.3.7.1. setFixture(SysteshFactory.eINSTANCE.createButonA());
    3.3.7.2. getFixture().activate();
3.3.8. Test outputs → activation signal is sent and sequence initiated
3.3.9. Test steps → Please refer to sequence diagram
3.3.10. Pass/fail → Passed

3.4. Case 4

3.4.1. Test case identification → TC-004
3.4.2. Test case revision history → V1.0 Initial version
3.4.3. ID of software under test → ButtonBTest.java
3.4.4. Test description → Button A functionality test
3.4.5. Requirements tested → HLR-2.1.1.3, HLR-2.1.1.8 , LLR-2.2.6
3.4.6. Test category → High and low level
3.4.7. Test inputs →
    3.4.7.1. setFixture(SysteshFactory.eINSTANCE.createButonB());
    3.4.7.2. getFixture().activate();
3.4.8. Test outputs → activation signal is sent and sequence initiated
3.4.9. Test steps → Please refer to sequence diagram
3.4.10. Pass/fail → Passed

3.5. Case 5

3.5.1. Test case identification → TC-005
3.5.2. Test case revision history → V1.0 Initial version
3.5.3. ID of software under test → CarLightTest.java
3.5.4. Test description → CarLight functionality test
3.5.5. Requirements tested → HLR-2.1.1.6, HLR-2.1.1.7, HLR-2.1.1.8, LLR-2.2.5
3.5.6. Test category → High and low level
3.5.7. Test inputs →
    3.5.7.1. setFixture(SysteshFactory.eINSTANCE.createCarLight());
    3.5.7.2. getFixture().setCar_current_light(Lights.GREEN);
    3.5.7.3. getFixture().switch_car_light();
3.5.8. Test outputs → CarLight switch from Green to Red
3.5.9. Test steps → Please refer to sequence diagram
3.5.10. Pass/fail → Passed

3.6. Case 6

3.6.1. Test case identification → TC-006
3.6.2. Test case revision history → V1.0 Initial version
3.6.3. ID of software under test → PedLightTest.java
3.6.4. Test description → Button A functionality test
3.6.5. Requirements tested → HLR-2.1.1.6, HLR-2.1.1.8, HLR-2.1.1.7, LLR-2.2.4
3.6.6. Test category → High and low level
3.6.7. Test inputs →
    3.6.7.1. setFixture(SysteshFactory.eINSTANCE.createPedLight());
    3.6.7.2. getFixture().setPed_current_light(Lights.RED);
    3.6.7.3. getFixture().switch_ped_light();
3.6.8. Test outputs → PedLight switch from Green to Red
3.6.9. Test steps → Please refer to sequence diagram
3.6.10. Pass/fail → Passed

3.7. Case 7

3.7.1. Test case identification → TC-007
3.7.2. Test case revision history → V1.0 Initial version
3.7.3. ID of software under test → Traffic Light System - Button initiated light change
3.7.4. Test description → System cycle test, from change request to light change
3.7.5. Requirements tested → HLR-2.1.1.1, HLR-2.1.1.2 , HLR-2.1.1.3, HLR-2.1.1.5, HLR-2.1.1.6, HLR-2.1.1.7, HLR-2.1.1.8, HLR-2.1.2.2, LLR-2.2.1, LLR-2.2.2, LLR-2.2.3, LLR-2.2.4, LLR-2.2.5, LLR-2.2.6
3.7.6. Test category → High and low level
3.7.7. Test inputs →
3.7.7.1. TestRunner.run(suite());
3.7.8. Test outputs → System works as expected
3.7.9. Test steps → Please refer to sequence diagram
3.7.10. Pass/fail → Passed

4. Test Procedure Descriptions

4.1. Junit test

4.1.1. ButtonATest
4.1.1.1. testActivate()
Test activate method
4.1.2. ButtinBTest
4.1.2.1. testActivate()
Test activate method
4.1.3. PedLightTest
4.1.3.1. testSwitch_ped_light()
Call switch_ped_light() then check if getPed_current_light() changed value or not
4.1.4. CarLightTest
4.1.4.1. testSwitch_car_light()
Call switch_car_light() the check if getCar_current_light() changed the value or not
4.1.5. TimerTest
4.1.5.1. testTick()
Call tick() function and check if time is incremented
4.1.5.2. testTimer_reset()
Not implemented

4.1.6. ControllerTest

4.1.6.1. testSwitch_lights_buttonA

Call switch_lights_buttonA() and call tick() 4 times, then check if isCar_traffic_active() is true or false, and that time is incremented

4.1.6.2. testSwitch_lights_buttonB

Same as previous test

4.1.6.3. testTimer_reset()

Not implemented

5. Coverage Report

5.1. HLR vs. Test Cases

5.1.1. HLR-2.1.1.1 is architectural - verified by review

5.1.2. HLR-2.1.1.4 is architectural - verified by review

5.1.3. HLR-2.1.1.9 is not tested or verified

5.1.4. HLR-2.1.1.10 is not tested or verified

5.2. Verification of functional requirements:

	HLR-2.1.1.2	HLR-2.1.1.3	HLR-2.1.1.5	HLR-2.1.1.6	HLR-2.1.1.7	HLR-2.1.1.8	HLR-2.1.2.1
TC-001	X						
TC-002			X				X
TC-003		X				X	
TC-004		X				X	
TC-005				X	X	X	
TC-006				X	X	X	
TC-007	X	X	X	X	X	X	X