**Smart traffic light**

**Software Design Document**

Created by:

**Netanel Davidov**

**Maxim Marmer**

Date: 16.3.2020

**Table of Contents**

1. **Introduction**
   1. Purpose
   2. Scope
   3. Overview
   4. Reference Material
2. **System Overview**
3. **System Architectural**
   1. Architectural Design
   2. Decomposition Description
   3. Design
4. **Data Design**
   1. Data Description
   2. Data dictionary
5. **Component Design**
6. **Human Interface Design**
   1. Overview of User Interface
   2. Screen Images and Actions
7. **Requirements Matrix**

**1. Introduction**

1.1 Purpose

This software design document describes the architecture and system design of Smart Traffic Light system (STL1.0). It will illustrate the appearance, the detailed structure of the components and complete description of the using the system. It is assumed that the reader has read the SRS, since this document also defines the implementation details of the desired behavior given the requirements within it.

1.2 Scope

This software is designed to improve work of traffic lights and can be used by the Ministry of Transport and Road Safety or the municipality. Improving the work of traffic lights will be carried out by collecting information and transmitting it in real time, which will lead to the correct distribution of traffic and reduce the waiting time on the roads. In the first version of the program, a model will be created to simulate the work of traffic lights on the basis of various traffic data, which will affect the change in traffic flow in the area between two crossroads. Our task will be to correctly distribute the work of the traffic light system so that the waiting time is reduced and the traffic lights work optimally.

1.3 Overview

This document is intended for describing user interface and design of the STL. Document describes missions of programs pages, their relationship with each other, description of each button and how client correctly have to use the system.

1.4 Definitions and Acronyms

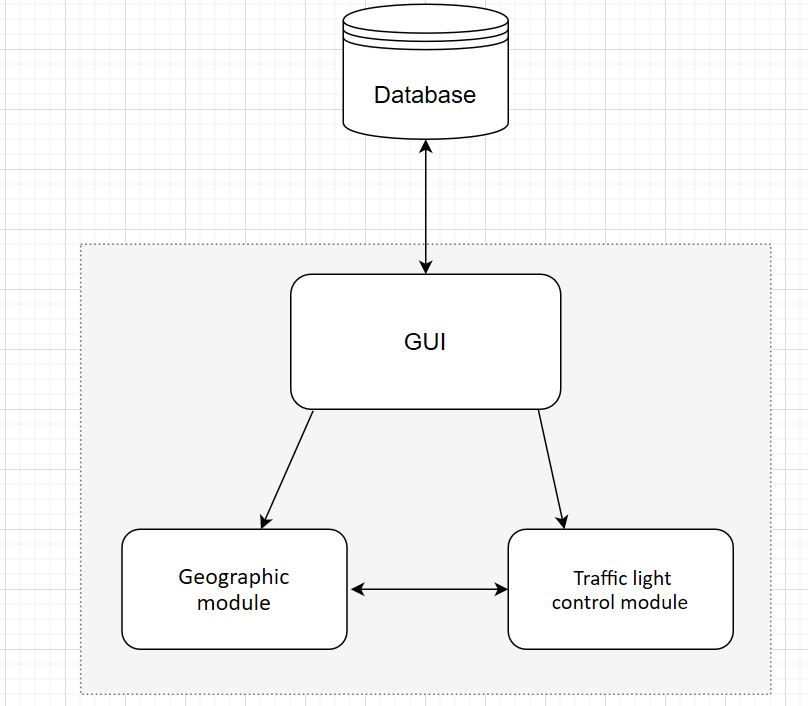
CSV: Comma Separated Values  
ID: Identifier   
GUI: Graphical User Interface   
XML: Extensible Markup Language  
SRS: Software Requirements Specification  
STL: Smart Traffic Light

**2. System Overview**

* The program consists of several windows and each window has its own unique purpose.
* The goal of the program is to graphically display traffic and solve the current situation on the road.
* The client at each time using the program can see one window that has its purpose.
* The user has access to move between windows forward and backward.
* To use the system, the client must familiarize themselves with the graphical interface described in section 6.2.
* All logic, algorithms and calculations of the program are not provided to the user, the client can only see the results and behavior of the program.
* The system has two types of users and the client must choose its own role.
* If the client is an analyst, then he will be able to manage the system and change the data, also he will have access to working with the results from the database, deleting and saving the results.
* If the client is an observer, then he gets access to work with existing results, but not their change.
* After choosing a situation, the user will be able to see with the help of graphical simulation how the system copes with the given task and changes the work of traffic lights.

**3. System Architecture**

3.1 Architectural Design



a. GUI - The graphical user interface, this subsystem allows to users to interact with the software through graphical icons and buttons. Its goal is to enhance the efficiency and ease of use for the underlying logical design of program, a design discipline named usability. Methods of user-centered design are used to ensure that the visual language introduced in the design is well-tailored to the tasks. The graphical user interface allows you to create a new simulation (which includes creating new nodes, adding cars, adding traffic lights, etc.) or opening an existing simulation by clicking buttons and icons that allow the user to do this clearly and visually. The reason for the decomposition of this system is due to the module, which means the main function of this subsystem is the graphical interface presented to the user.

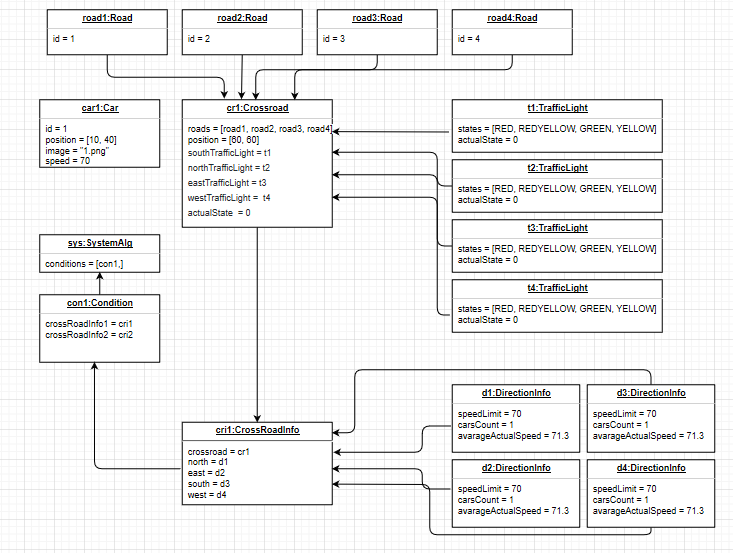
b. Crossroad control module - This subsystem allows control of switching and monitoring of traffic lights colors according to road data, amounts of cars, car speed, loads, lanes, number of crossroads, etc. This subsystem is responsible for the complete set of traffic lights in the current simulation and is responsible for performing the algorithmic calculations that determine the traffic light exchange times most efficiently.

c. Geographic module - This subsystem holds the network of traffic crossroads, traffic lights, roads and cars. In each simulation, a system must create such a network that represents different locations of these objects. This subsystem helps the general system to imitate as much as possible the realistic model of traffic roads.

d. Database - In each simulation, it is possible to insert new data when it is saved when running the simulation in the database and also the possibility to extract historical data from the database. These data are: locations - traffic crossroads, traffic lights, roads, cars; Traffic light colors; Times; Algorithms details etc. This subsystem allows the general system to conveniently access the database and retrieve / store the desired data.

3.2 Decomposition Description

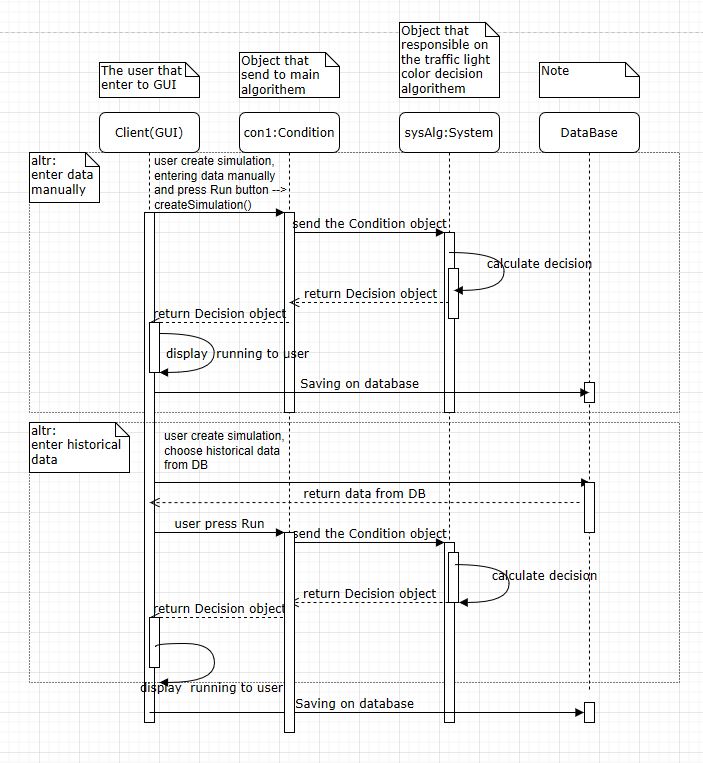
First to explain the details of decomposition of crossroad control module and geographic module (subsystems), we will show the appropriate **object diagram** that describe the Integration of this two modules:

  
  
  
When simulation of any kind is created (details were manually entered by the customer or details were retrieved from the database or details were randomly entered) the objects described above are created, you can see a real example of the objects that are created in the above object diagram. In addition, this diagram describes the objects created after the client clicked Run and before the Condition object was created, which is designated as a parameter for the class of complex algorithms.

The object that responsible algorithms

The following diagram shows the "Sequence diagram" that shows what happens in the entire software from the beginning of the simulation to the moment the visual process starts running and what happens after the condition object is created for the algorithm class.

This chart shows the importance of separating the database and GUI modules and decomposing them:



3.3 Design Rationale

In section 3.1, we discussed various architectures that we finally chose for this architecture as it fulfills our rationale (of the system you want to build). The main point of our system is to simulate smart traffic lights at intersections that are under the control of a general system. Dismantling to subsystems has helped us better model our system and focus on each part individually, finally reaching a complete system that works together with all subsystems fully.

As part of our discussion, we wanted to focus primarily on what the system should provide in its initial version and so we focused on the main goal of creating a smart traffic light simulation that knows how to change their color according to road conditions and external control that controls them, so that each intersection will be the most efficient cars crossing.

**4. Data Design**

4.1 Data Descriptions

The application information area is in the Select conditions window. Analyst has all permissions to change data in this window. Observer has same window but cannot change conditions manually, observer has permissions to use only exists databases files.

Before starting the simulation, the client must select the conditions. There are the following options for this.

* set manually
* choose randomly
* download from database

After the client selects the data entry method and enters the necessary conditions, all the data will be turned into objects.

The client must enter the following data for two crossroads in four directions:

* number of cars
* allowed speed
* true speed

This data creates the CrossroadInfo object. Using the method setCrossroadInfo the system will try to use the entered data and will return answer, if the answer is true, then the data was correct, otherwise the answer will be false. If the data was incorrect, the client will see a warning from the system, in another case, the object will be created and the user will be able to continue.

In order to load data from the database, the system produces similar actions. We get information from the database by using Database object that has method getFromDatabase which return to the system data from tables in string type that will converted to suitable Objects.

The system stores data in the following tables:

Изображение выглядит как снимок экрана

Автоматически созданное описание

4.2 Data Dictionary

Objects:

**Car** (Implements DrawObject)

Attributes:

* Image image – image of car

Functions:

* Image getImage () – return image of car
* Int generateRandomNumber () – generate random number in range 1-6

**Conditions**

Attributes:

* CrossroadInfo crossroadInfo1 – information of first crossroad
* CrossroadInfo crossroadInfo2 – information of second crossroad

Functions:

* Conditions (CrossroadInfo info1, CrossroadInfo info2) – constructor, set crossroads.
* CrossroadInfo getCrossroadInfo1 () – return crossroadInfo1
* CrossroadInfo getCrossroadInfo2 () – return crossroadInfo2

**Crossroad**

Attributes:

* TrafficLight southTrafficLight - traffic light towards south
* TrafficLight northTrafficLight - traffic light towards north
* TrafficLight eastTrafficLight - traffic light towards east
* TrafficLight westTrafficLight - traffic light towards west
* Road [] roads – array of roads that belongs to crossroad
* Int CROSSROAD\_SIZE = 4 – constant value of roads amount
* Int actualState = 0 – state of crossroad. Each crossroad has 8 states that explain states of traffic lights to each direction. Traffic signals varies in pairs:

|  |  |  |
| --- | --- | --- |
| State | Directions: **South** and **North** | Directions: **East** and **West** |
| 0 | Red | Red |
| 1 | Red Yellow | Red |
| 2 | Green | Red |
| 3 | Yellow | Red |
| 4 | Red | Red |
| 5 | Red | Red Yellow |
| 6 | Red | Green |
| 7 | Red | Yellow |

Functions:

* Void changeState () – change states of traffic lights
* Void addNorthRoad (Road north) – add north road to array of roads that crossroad includes
* Void addSouthRoad (Road south) – add south road to array of roads that crossroad includes
* Void addEastRoad (Road east) – add east road to array of roads that crossroad includes
* Void addWestRoad (Road west) – add west road to array of roads that crossroad includes

**CrossroadInfo**

Attributes:

* Crossroad crossroad – the crossroad to which the information belongs
* DirectionInfo north – north traffic information
* DirectionInfo east – east traffic information
* DirectionInfo south – south traffic information
* DirectionInfo west – west traffic information

Functions:

* Crossroad getCrossroad () – return the crossroad
* Boolean setCrossroadInfo (String [] carsCount, String [] speedLimit, String [] actualSpeed) – try to set conditions into DirectionInfo objects. Return true if data was correctly and set operation was success, otherwise return false.
* Boolean checkData (String [] data) – Return true if all values of data array can be converted to integer, otherwise return false.
* Boolean isInt (String value) – return true if the value can be converted to integer.
* DirectionInfo getNorth () – return north DirectionInfo
* DirectionInfo getEast () – return east DirectionInfo
* DirectionInfo getSouth () – return south DirectionInfo
* DirectionInfo getWest () – return west DirectionInfo

**Database**

Functions:

* Conditions getFromDatabase (String id) – return Conditions by id from database.
* Void putToDatabase (Conditions conditions) – save to database conditions of last run

**DirectionInfo**

Attributes:

* Int carsCount – amount of cars
* Int speedLimit – speed limit
* Int actualSpeed – current traffic speed

Functions:

* DirectionInfo (String carsCount, String speedLimit, String actualSpeed) – constructor, convert arguments to integer type and set them into class variables.
* Int parseToInt(String value) – parse value to integer and return it.
* Int getCarsCount () – return carsCount variable.
* Int getSpeedLimit () – return speedLimit variable.
* Int getActualSpeed () – return actualSpeed variable.

**Road**

Attributes:

* Int ID – road number

Functions:

* Road (int id) – constructor with argument that will be used as ID
* Int getRoadId () – return road number

**TrafficLight**

Attributes:

* States [] STATES – array of traffic light state
* Int actualStates – value that store index of actual state

Functions:

* TrafficLight () – constructor
* Void changeState () – change states of traffic light
* String getActualStateName () – return name of actual state
* Image getActualStateImage () – return Image of actual state

**State**

Abstract class that implements DrawObject interface

**GreenState** (Extends State)

Attributes:

* String name – name of state

Functions:

* Image getImage () – return image of green signal
* String getName () – return name of state

**YellowState** (Extends State)

Attributes:

* String name – name of state

Functions:

* Image getImage () – return image of yellow signal
* String getName () – return name of state

**RedState** (Extends State)

Attributes:

* String name – name of state

Functions:

* Image getImage () – return image of red signal
* String getName () – return name of state

**RedYellowState** (Extends State)

Attributes:

* String name – name of state

Functions:

* Image getImage () – return image of red-yellow signal
* String getName () – return name of state

**5. Component Design**

After the user has filled in all the fields and decided to continue, he clicks the run button. After clicking on it, the process of creating system objects occurs, as shown in the figure.

Изображение выглядит как объект, часы

Автоматически созданное описание

This program operation is described below in the pseudo code:

Explanation of variables:

* carsCount – array of data entered by the user in GUI
* speedLimit - array of data entered by the user in GUI
* actualSpeed - array of data entered by the user in GUI
* crossroadInfo – object that includes information about the crossroad   
  (can see the definition in section 4.2)
* system – object that execute actions for solving the current traffic situation
* id – identification key of chosen conditions by user from the database
* data – uploaded conditions from the database

Run (carsCount, speedLimit, actualSpeed):

1 crossroadInfo1, crossroadInfo2

2 if setCrossroadInfo (carsCount, speedLimit, actualSpeed)

3 conditions (crossroadInfo1, crossroadInfo2)

4 Start (system)

5 if-else

6 sendFailMessage(“Data is incorrect”)

7 end-if

If the user has loaded the conditions from the database and decided to continue, he must click the start button. After clicking on it, the process of creating system objects occurs, as shown in the figure.

Изображение выглядит как объект

Автоматически созданное описание

This program operation is described below in the pseudo code:

data = getFromDatabase()

Run (data):

1 crossroadInfo1, crossroadInfo2

2 carsCount, speedLimit, actualSpeed = splitDataFromDatabase(data)

3 if setCrossroadInfo (carsCount, speedLimit, actualSpeed)

4 conditions (crossroadInfo1, crossroadInfo2)

5 system = setConditions (conditions)

6 Start (system)

7 if-else

8 sendFailMessage (“Data is incorrect”)

9 end-if

setCrossroadInfo (carsCount, speedLimit, actualSpeed):

1 answer = checkData (carsCount) & checkData (speedLimit) & checkData (actualSpeed)

2 if answer

3 north, east, south, west = setData (carsCount, speedLimit, actualSpeed)

4 return true

5 end-if

6 return false

After the system has received conditions that the user can run the simulation and save the result:

Start (system):

1 conditions = getConditions(system)

2 result = algorithm (conditions)

3 simulation (result)

4 if save

5 saveResult (result)

6 end-if

SaveResult (result):

1 data = arrangeResultToDatabaseFormat (result)

2 putToDatabase (data)

**6. Human Interface Design**

6.1 Overview of User Interface

The client of the system will be able to work with the program through a graphical interface. The GUI of the system makes it possible to select the type of client, adjust the road conditions and track changes when performing the system algorithms. Significant user actions are accompanied by a repeated request from the system, if the user made a mistake in entering data, the system will notify him of errors, so the client will be able to cancel his actions by pressing the corresponding button.

The main task of the user in the system is to choose certain road conditions that he wants to analyze in order to see the efficiency of the STL system. In order to change the conditions of the road, the user must enter the "Choose traffic conditions" screen and select the conditions he needs manually, randomly or download from the database. The basic information when working with road conditions that the STL system uses is the number of cars, the speed limit and the current speed on the road. The client will be able to manually enter this data in the appropriate field, and they will later be converted into objects. The program provides access to the following data fields:

* Route - direction of travel
* Cars counter - the number of cars that the system must process
* Speed limit - speed limit in the selected direction
* Actual speed - current speed in the selected direction

These fields are duplicated twice for each crossroad so the client will have to enter them separately.

After the client has chosen the conditions of road traffic, the system will check the correctness of the entered data and, if successful, the system will continue to work, otherwise the user will need to change the data.

The screens layout is as followsИзображение выглядит как снимок экрана

Автоматически созданное описание

6.2 Screen Images and Actions

Home page – this page for user greetings and version information

Изображение выглядит как снимок экрана

Автоматически созданное описание

* Clicking on the “About Us” button will open next window.

Изображение выглядит как снимок экрана

Автоматически созданное описание

* Clicking on the “Let’s start” button will open the “Client type” window.

Изображение выглядит как снимок экрана

Автоматически созданное описание

* The user must select his role, after which the "Traffic conditions" window will open.

Изображение выглядит как снимок экрана

Автоматически созданное описание

In this window user have to fill all conditions of traffic and after this click to the “Run” button.

* Clicking on the “Random” button will open a confirm window. “Yes” to choose conditions randomly, “No” to cancel action.

Изображение выглядит как снимок экрана

Автоматически созданное описание

* Clicking on the “Reset” button will open a confirm window. “Yes” to reset all conditions to the initial, “No” to cancel action.

Изображение выглядит как снимок экрана

Автоматически созданное описание

* Clicking on the “Back” button will open a confirm window. “Yes” to go back, “No” to cancel action.

Изображение выглядит как снимок экрана

Автоматически созданное описание

* Clicking on the “Database” button will open window with saved data from database.
* Clicking on the “Info” button will open a window with instruction how fill the fields.
* Clicking on the “Close” button will open a confirm window. “Yes” to close system, “No” to cancel action.

Изображение выглядит как снимок экрана

Автоматически созданное описание

* Clicking on the “Run” button will open the “Simulation” window.

Изображение выглядит как снимок экрана

Автоматически созданное описание

* Clicking on the “Back” button will open a confirm window. “Yes” to go back, “No” to cancel action.

Изображение выглядит как снимок экрана

Автоматически созданное описание

* Clicking on the “Start” button will start the simulation.
* Clicking on the “Save” button will open the form for save the results and conditions.

**7. Requirements Matrix**

|  |  |  |
| --- | --- | --- |
| SRS phase | SDD phase | Satisfaction |
| Traffic Light | TrafficLight | Partially |
| Crossroad | Crossroad | Partially |
| Road | Road | Partially |
| Vehicle | Car | Partially |
|  | CrossroadInfo | Absent |
|  | DirectionInfo | Absent |
|  | GreenState | Absent |
|  | YellowState | Absent |
|  | RedState | Absent |
|  | RedYellowState | Absent |
|  | Conditions | Absent |
|  | Database | Absent |

More detailed description of system components can be found in paragraph 4.