



CmpE 275 – Spring 2010

Under the Guidance of
Prof. Dr. Jerry Gao

Project Report on
Component Based Elevator Simulation System

Submitted By

Aarthi Varadarajan (006543498)
Jahnavi Athuluru (006481124)
Sandhya Rajagopalan (006983171)

05/12/2010

Acknowledgements

*We would like to **thank** our **Professor Dr. Jerry Gao** for his incredible guidance which helped us in the successful completion of our project.*

Contents

1. Introduction	4
2. Objectives.....	4
3. Notations.....	5
4. System Architecture.....	5
5. Configurable parameter description.....	7
6. Component Composition Diagram	8
7. Component Interaction Diagram	9
8. Component Behavior Specification.....	10
8.1 Car State Transition Diagram	11
8.2 Door State Transition Diagram	11
8.3 UserPanel State Transition Diagram	12
9. Collaboration diagram	12
10. Component Design.....	13
10.1 Low-Level Class Diagram.....	13
11. Component API design.....	15
12. Component Implementation	17
13. Screen shots	19
14. Comparison with existing system	24
15. Future Directions	24
16. Conclusion and Learning	24
17. Reference	25

1. Introduction

Software systems today are becoming more and more large-scale, complex and uneasily controlled. This results in high development cost, low productivity, unmanageable software quality and high risk to move to new technology. Consequently, there is a growing demand of searching for a new, efficient, and cost-effective software development paradigm. One of the most promising solutions today is the component-based software development approach [1].

A component is an individual module that encapsulates a set of related data or functions. Components communicate with each other via *interfaces*. When a component offers services to the rest of the system, it adopts a *provides* interface which specifies the services that can be utilized by other components.

2. Objectives

The main objectives of the component based elevator system are as follows:

- To develop individual components that can be easily configured, deployed and executed independently.
- To be able to easily replace existing components with new components.
- To assemble individually developed components and make them work together as a system.
- To allow the user to mix and match different components to form a system.
- To design well defined interfaces so that individual components can be extended and enhanced without affecting the working of the system as well as other components

3. Notations


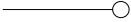


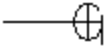
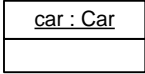
Notations	Meaning
	Represents a component node
	Represents a <i>provides</i> interface for a component
	Represents a <i>requires</i> interface for a component
	Represents dynamic composition
	Represents static composition
	Represents instance of a component

Table 1: Diagram Notations

4. System Architecture

Given below is a high level view of the components in our elevator simulation system and also their “*requires*” and “*provides*” interfaces.

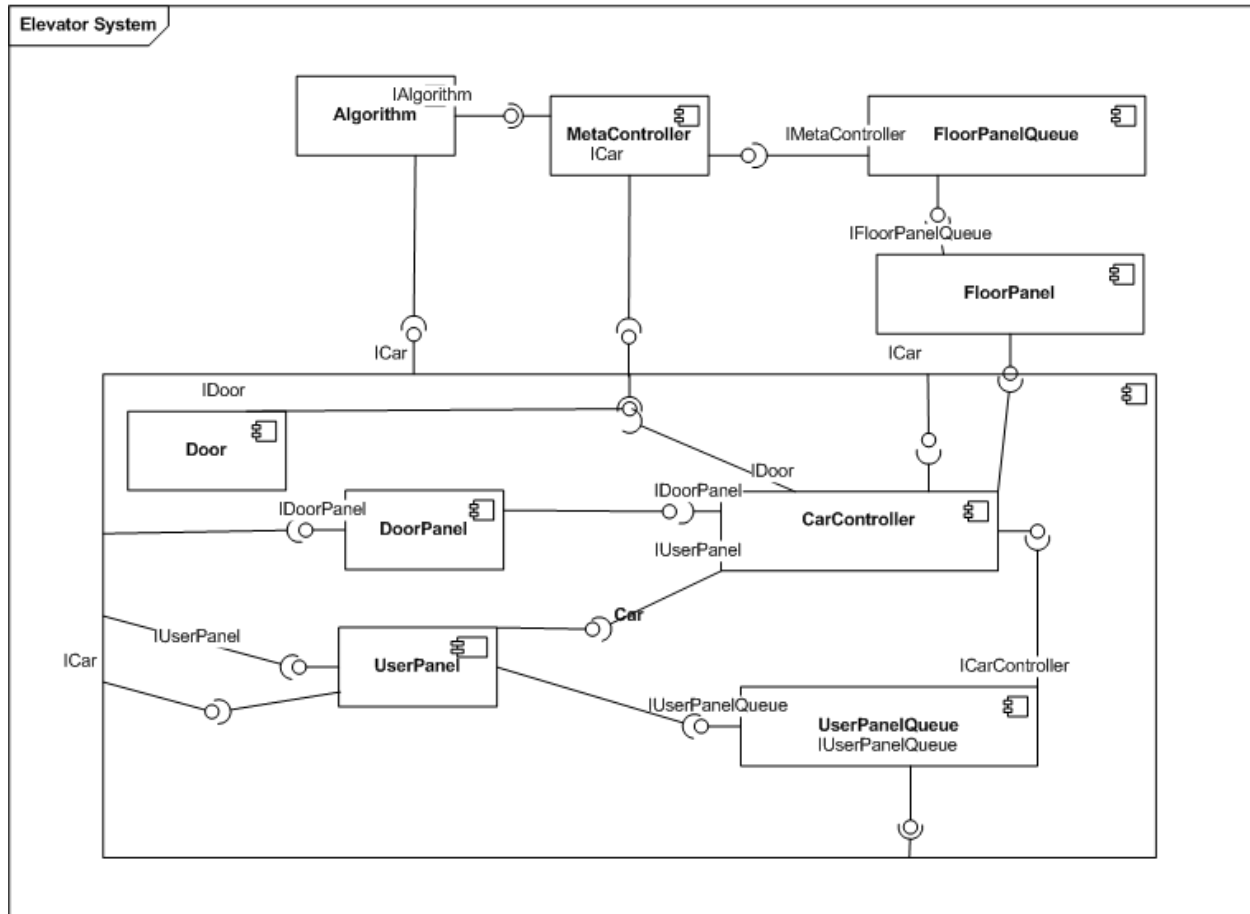


Figure 1: Component Diagram for Elevator System

Given below is a description of components used in the system.

Component	Description
UserPanel	Represents the user panel of an elevator system. User interacts with this component to give floor requests. It consists of floor buttons starting from 1 to maximum number of floors.
Door	Represents door of an elevator system.
DoorPanel	Represents door panel of an elevator system. User interacts with this component to send open/close door requests. It has the open and close buttons
UserPanelQueue	Queue components that queues, sorts and de-queues floor requests given by user. It also dispatches the requests to the CarController component.

CarController	Backend component that controls moving of car and processing door requests.
Car	A component that houses all the above components.
FloorPanel	Represents floor panel of an elevator. User interacts with this component to send direction and floor requests. It has the up/down buttons.
FloorPanelQueue	Component that queues and de-queues floor panel requests. It also dispatches requests to the MetaController component
Algorithm	Component that computes which car has to be chosen to process requests receives from floor panel
MetaController	Component that controls all Car and CarController components.

Table 2. List of components

5. Configurable parameter description

As one of the goals of this system is to develop a configurable elevator system, this section presents what are the configurable parameters for an elevator system on the whole and also configurable parameters for individual components:

Elevator System: Number of cars, number of floors, door type to be used for cars, user panel and floor panel button color, user panel and floor panel button color when pressed, type of display for floor panel and door panel and algorithm to be used to select a car for processing floor requests.

Car: Type of user panel, door, door panel, car controller and user panel queue component to be assemble to form a car.

UserPanel: Number of floors, user panel button color and type of car.

Door: Type of door – single/double.

DoorPanel: Display type of buttons, type of car

UserPanelQueue: Type of car or car controller.

CarController: Type of car it controls

FloorPanel: Floor panel queue into which it places the requests.

MetaController: Algorithm to be used.

6. Component Composition Diagram

a. Static Component composition

Composition relationships are strong form of containment. It is depicted in the UML using a class relationship. The static view of the composition is shown below.

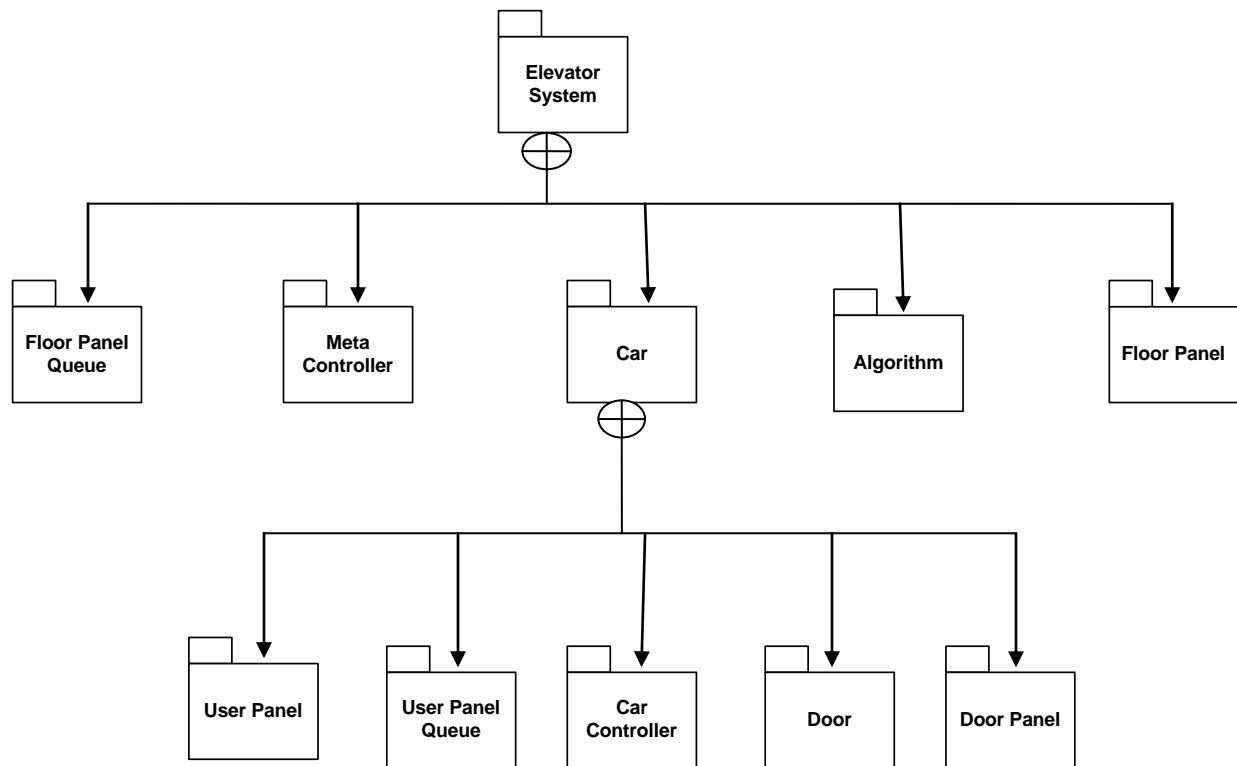


Figure 2: Static Component composition diagram for Elevator system.

b. Dynamic Component Composition

The dynamic composition for the elevator system is shown in the figure below

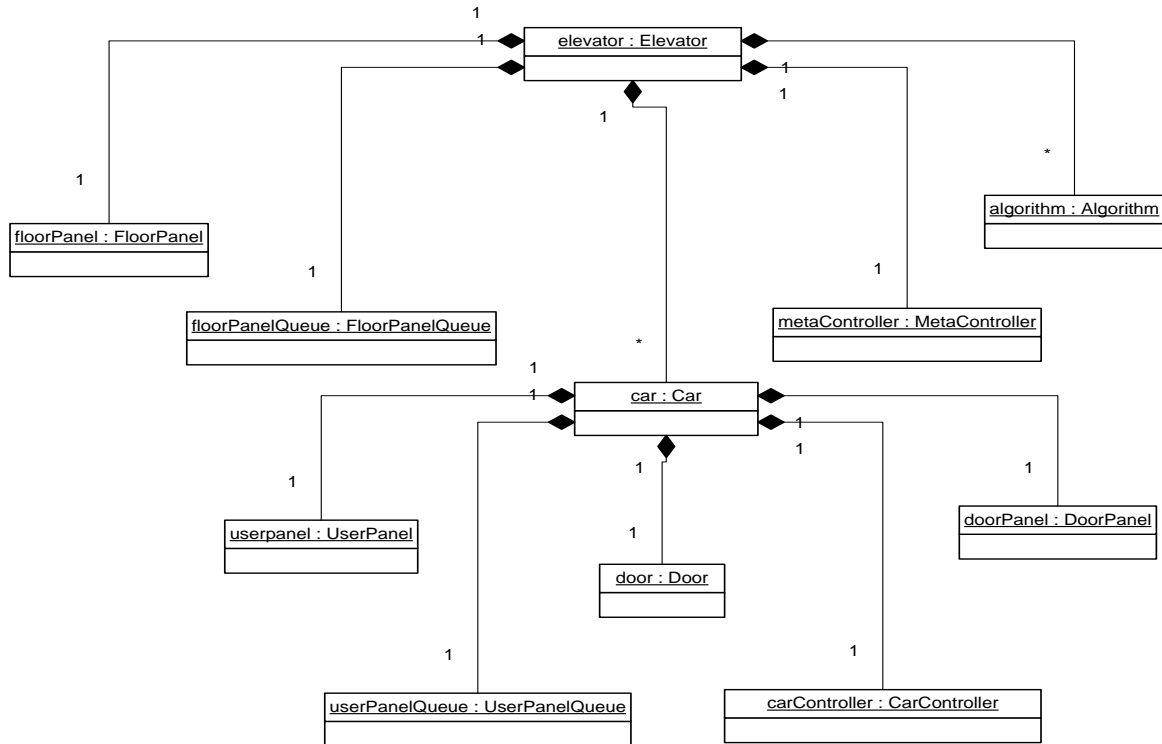


Figure 3: Dynamic component composition of the Elevator System

7. Component Interaction Diagram

Interaction diagrams model the behavior of use cases by describing the way groups of objects interact to complete the task. Interaction diagrams are used when you want to model the behavior of several objects in a use case. They demonstrate how the objects collaborate for the behavior.

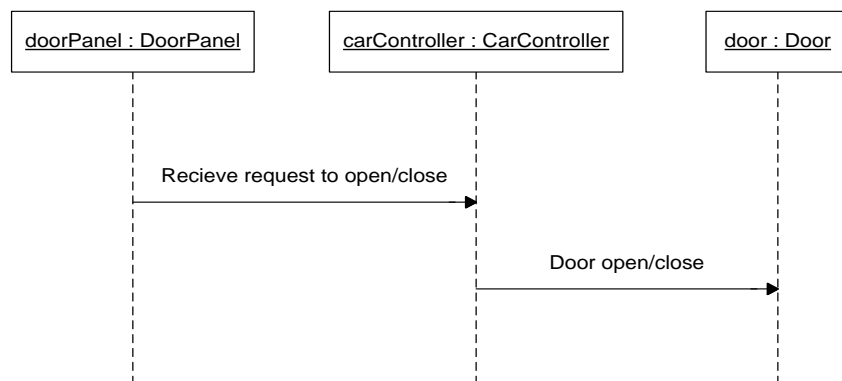


Figure 4: Interaction diagram for Door component

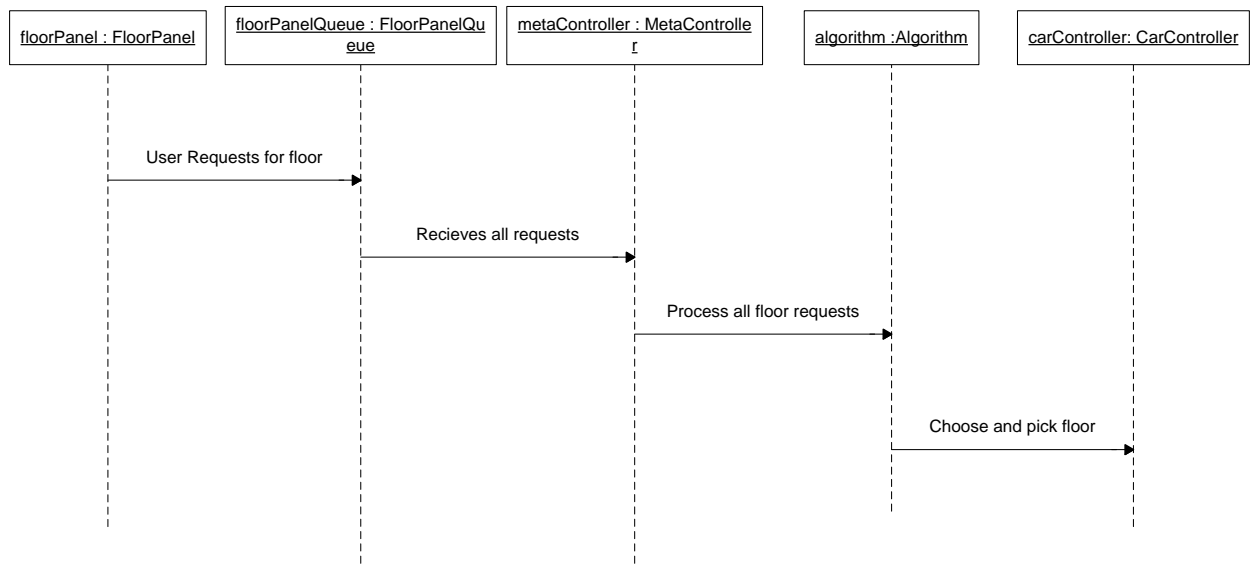


Figure 5: Interaction diagram for FloorPanel component

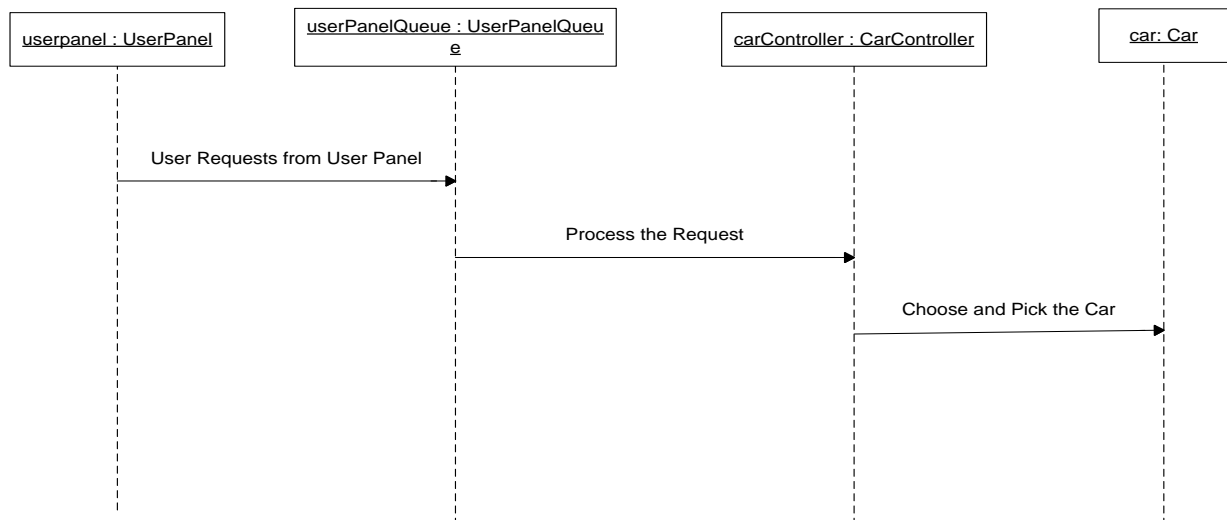


Figure 6: Interaction diagram for UserPanel component

8. Component Behavior Specification

Behavior specification depicts the states and the events causing transition between states for a component. In this system we have modeled the states of car, user panel and door component which is shown below.

8.1 Car State Transition Diagram

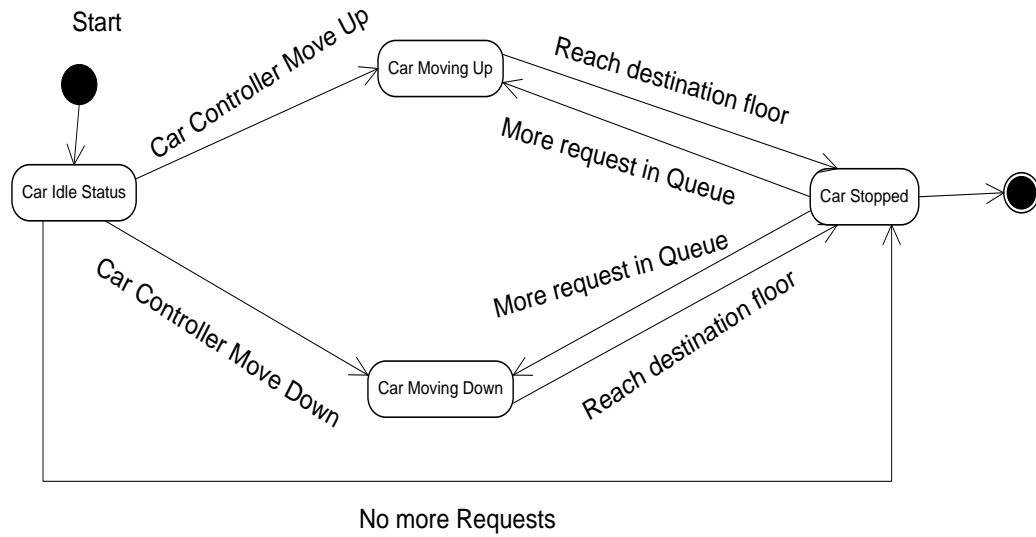


Figure 7: State Transition Diagram for Car component

8.2 Door State Transition Diagram

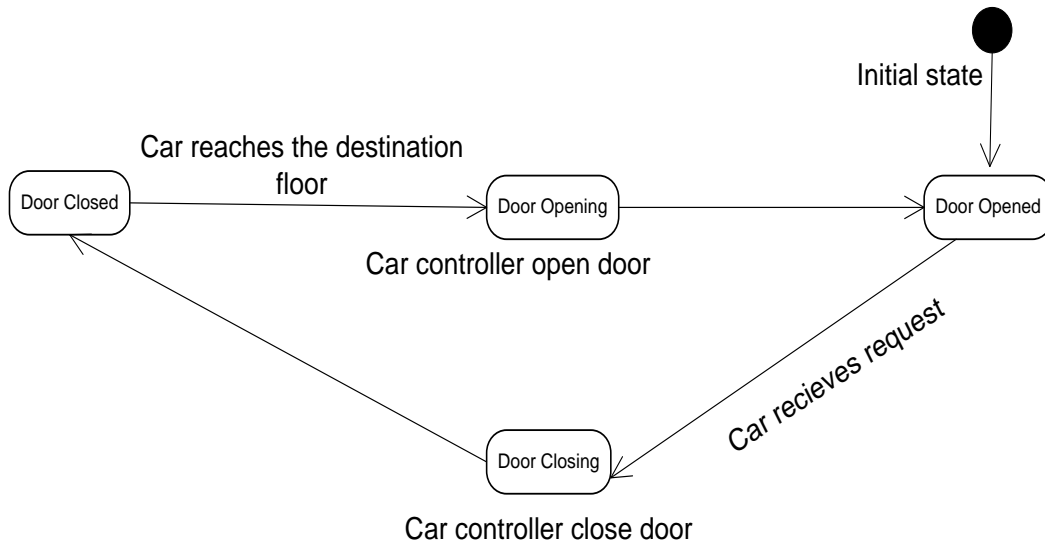


Figure 8: State Transition Diagram for user Door component

8.3 UserPanel State Transition Diagram

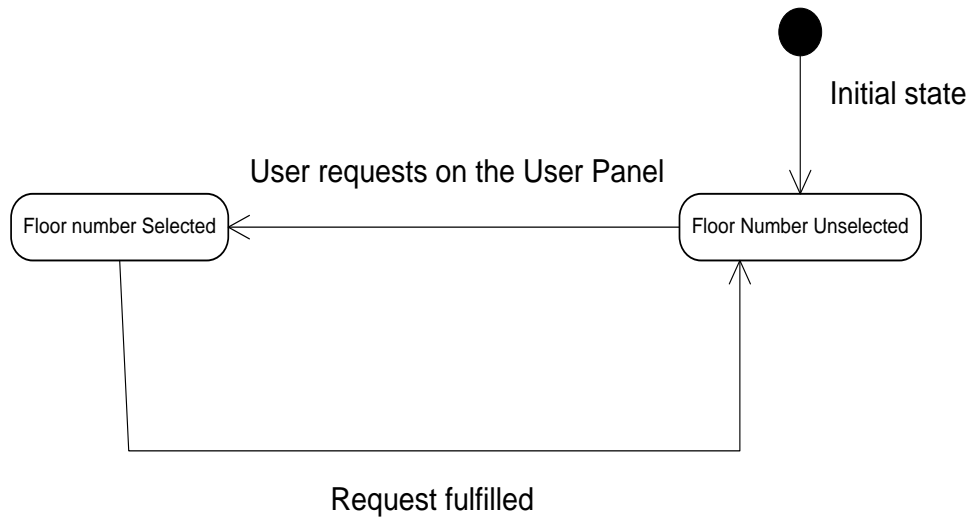


Figure 9: State Transition Diagram for user panel component

9. Collaboration diagram

Request for elevator from floor

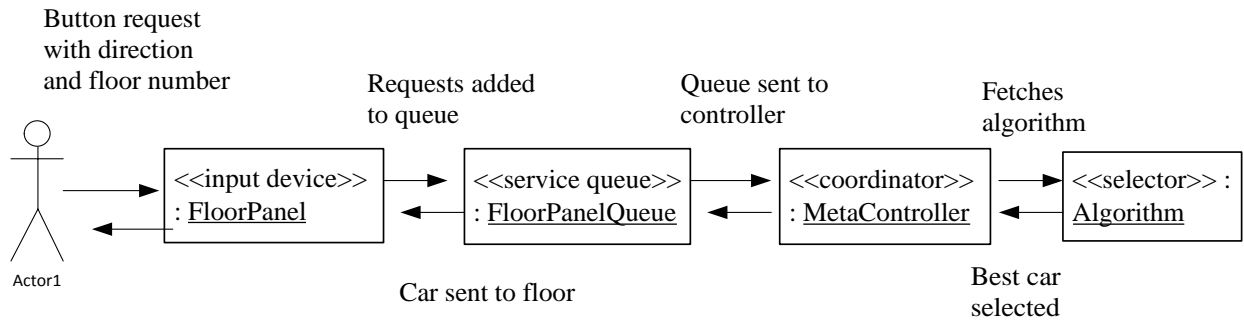


Figure 10: State Transition Diagram for user panel component

Request for floor from elevator

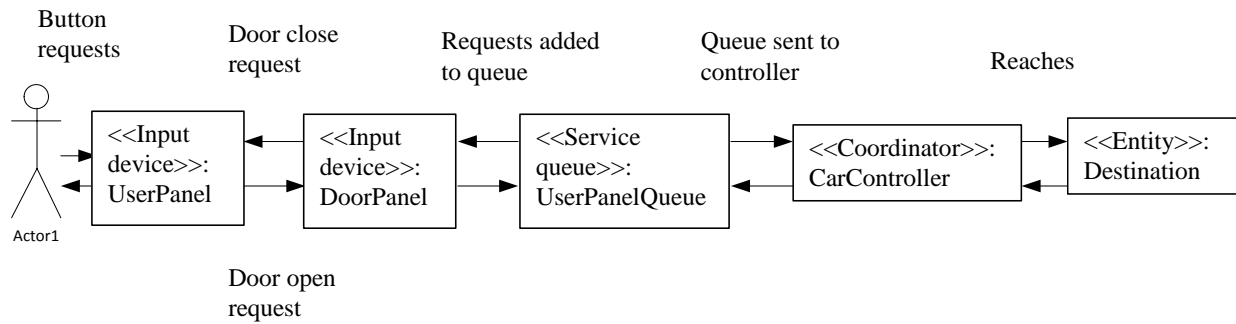


Figure11: State Transition Diagram for user panel component

10. Component Design

10.1 Low-Level Class Diagram

Given below is the low level class diagram for the entire elevator system.

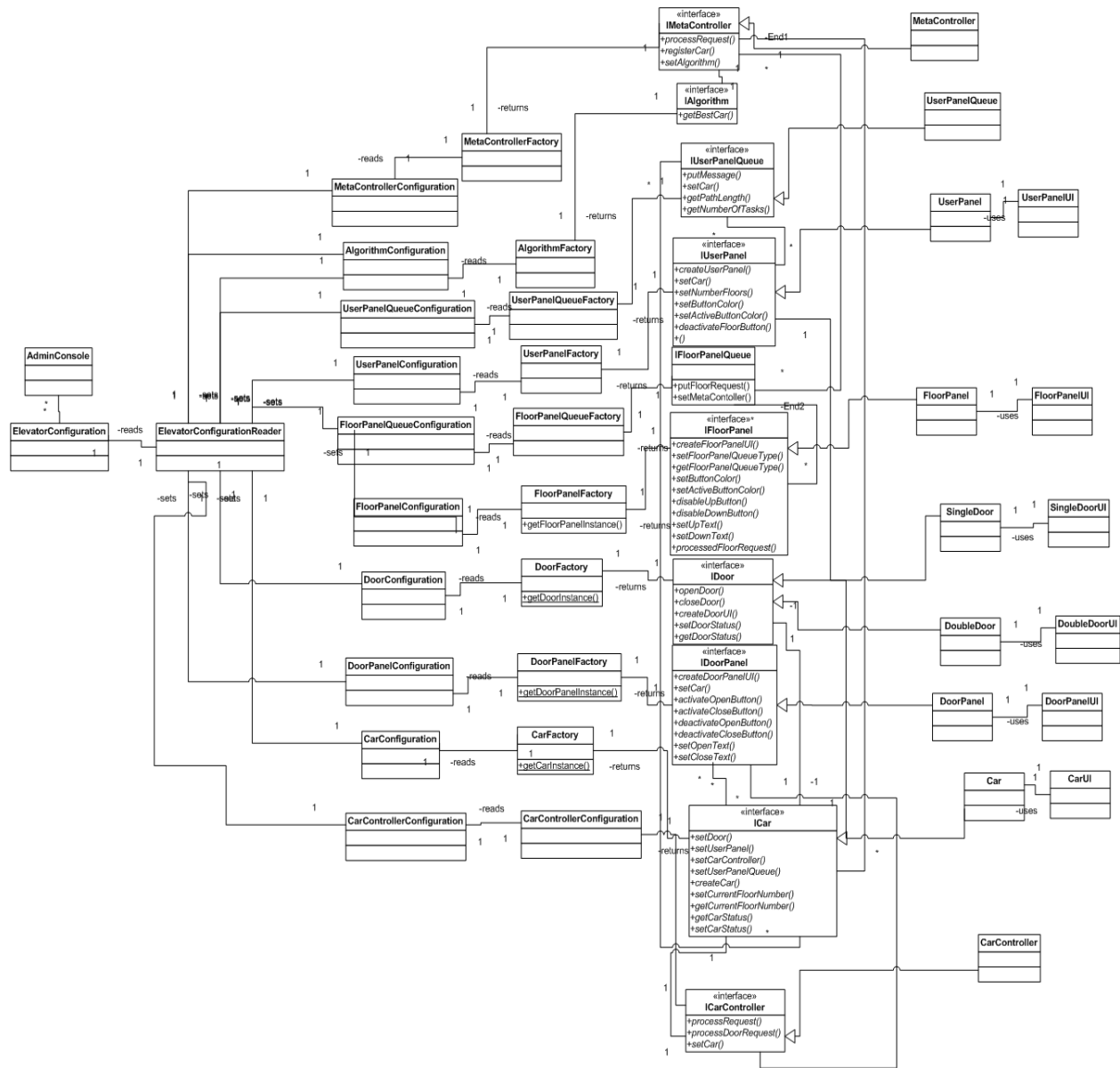


Figure 12. Low-Level Class Diagram for Elevator System

Every component consists of the following:

- A configuration class that has the configurable parameters for the component
- A factory class that is used to instantiate specific instances for a component by reading the configuration class.
- A component interface that is being exposed and used by other components.

Given below is another view that specifies the detailed class diagram at component level for components Car, UserPanel and Door. This view mainly focuses on the class design for each

component. As we see that classes for a component are encapsulated within the component and a component only exposes its interface to other components.

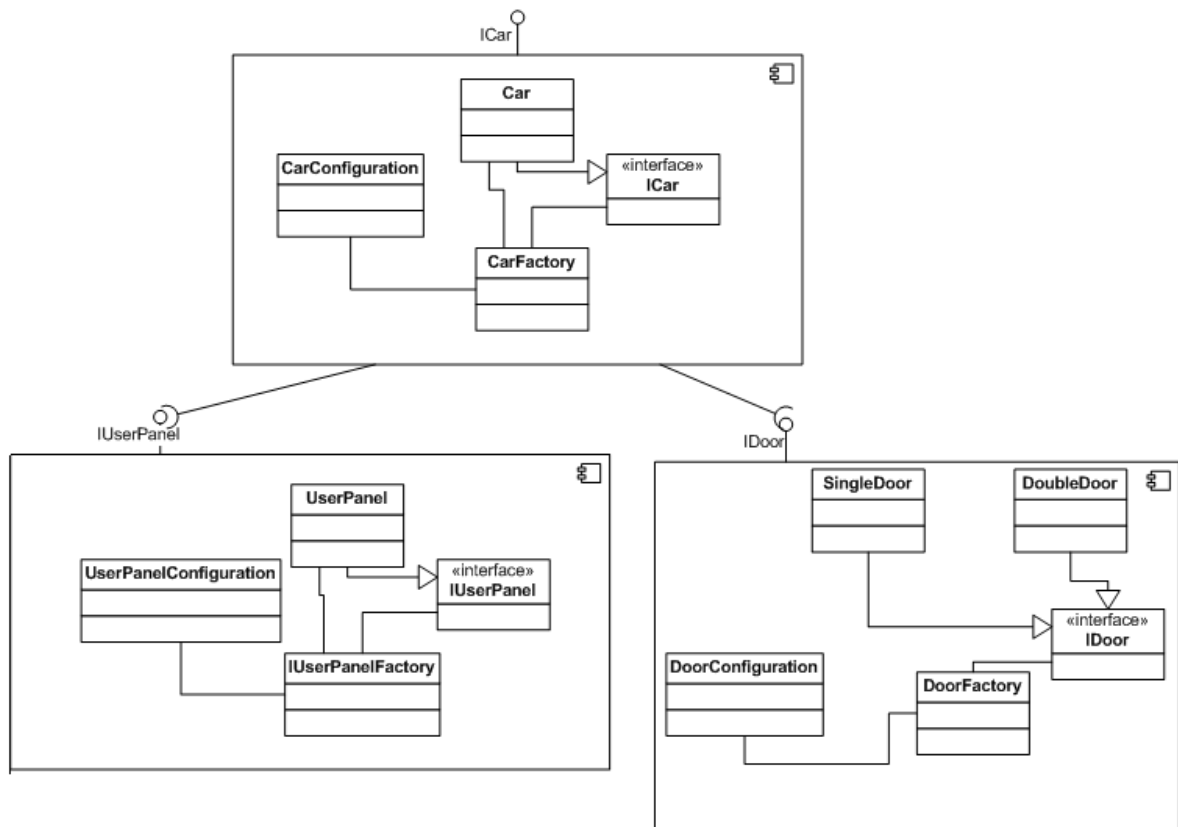


Figure 13. Detailed Class diagram at component level

11. Component API design

The table describes what all each component can provide and what they require. A component can provide functionality or in other words can be deployed using its interface, which other components can use at the same time a component will require configuration with other components.

	Requires	Provides
Car	iUser panel iDoor iDoorPanel	iCar

	iUserPanelQueue iCarController	
User panel	iCar iCarController iDoor iUserPanel iUserPanel Queue	iUserPanel
Door	iCar iCar Controller iDoor iUserPanel iUserPanelQueue	iDoor
Door panel	iCar iCar Controller iDoor iDoorPanelQueue	iDoorPanel
CarController	iCar iDoor iFloorPanel iDoorPanel iUserPanel	iCarController
User Panel queue	iCar iCar Controller	iUserPanelQueue

Floor panel	iFloorPanel Queue	iFloorPanel
Meta controller	iAlgorithm iCar iCar Controller iUserPanel Queue	iMetaController
Algorithm	iCar iCar Controller iUserPanel Queue	iAlgorithm
Floor Panel Queue	iMeta controller	iFloorPanelQueue

Table 3: Overview of Components with provides and requires interface

12. Component Implementation

Flow of the system

1) Admin – The admin can choose color, button types (whether default/ symbolized), no of cars to operate, no of floors.

Configuration – The above can be configured using Configuration classes for each component.

Factory – Based on the configuration file, factory class will create a new object

Implementation – Interface – New objects created will implement its interface where data and functions are defined.

User-Interface – This part consists of the Graphical user interface for the user or admin to operate the elevator system.

2) User -

2.1) Request for car from floor (FloorPanel) – When a car is requested from floor, requests containing direction and current floor value are put in queue and sent to the MetaController which selects best car using algorithm.

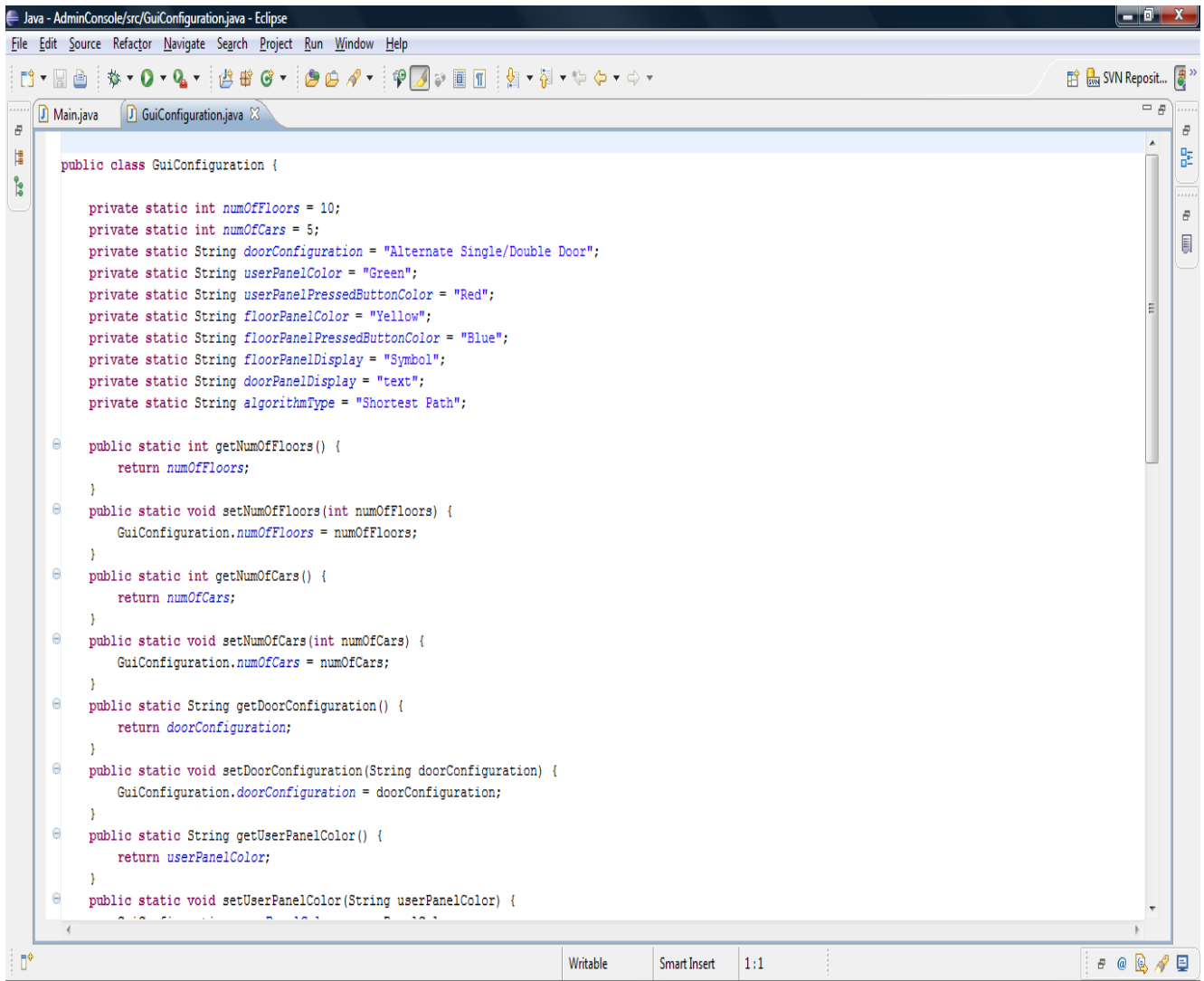
2.2) For requests coming for floor from inside the car (UserPanel) – This is done through user panel and queue of requests containing destination floor numbers and direction value is sent the CarController which processes every floor request in the queue.

2.3) Open or close door (DoorPanel) - The user can then operate the door panel using which the door of the car can be opened or closed. Requests are taken in DoorQueue and transferred to Door which opens or closes.

3) Conventions

- All Interfaces are named starting with I for clarity and to avoid ambiguity.
- Enumerators are used for better understandability.
- Usage of Getter and Setter methods followed through out.
- Factory pattern to create a new object and to generalize the basics properties of the object with the configuration settings given by the admin.
- Every component has a main function which means it can be used independently.

13. Screen shots



```
public class GuiConfiguration {

    private static int numOffFloors = 10;
    private static int numOfCars = 5;
    private static String doorConfiguration = "Alternate Single/Double Door";
    private static String userPanelColor = "Green";
    private static String userPanelPressedButtonColor = "Red";
    private static String floorPanelColor = "Yellow";
    private static String floorPanelPressedButtonColor = "Blue";
    private static String floorPanelDisplay = "Symbol";
    private static String doorPanelDisplay = "text";
    private static String algorithmType = "Shortest Path";

    public static int getNumOffFloors() {
        return numOffFloors;
    }

    public static void setNumOffFloors(int numOffFloors) {
        GuiConfiguration.numOffFloors = numOffFloors;
    }

    public static int getNumOfCars() {
        return numOfCars;
    }

    public static void setNumOfCars(int numOfCars) {
        GuiConfiguration.numOfCars = numOfCars;
    }

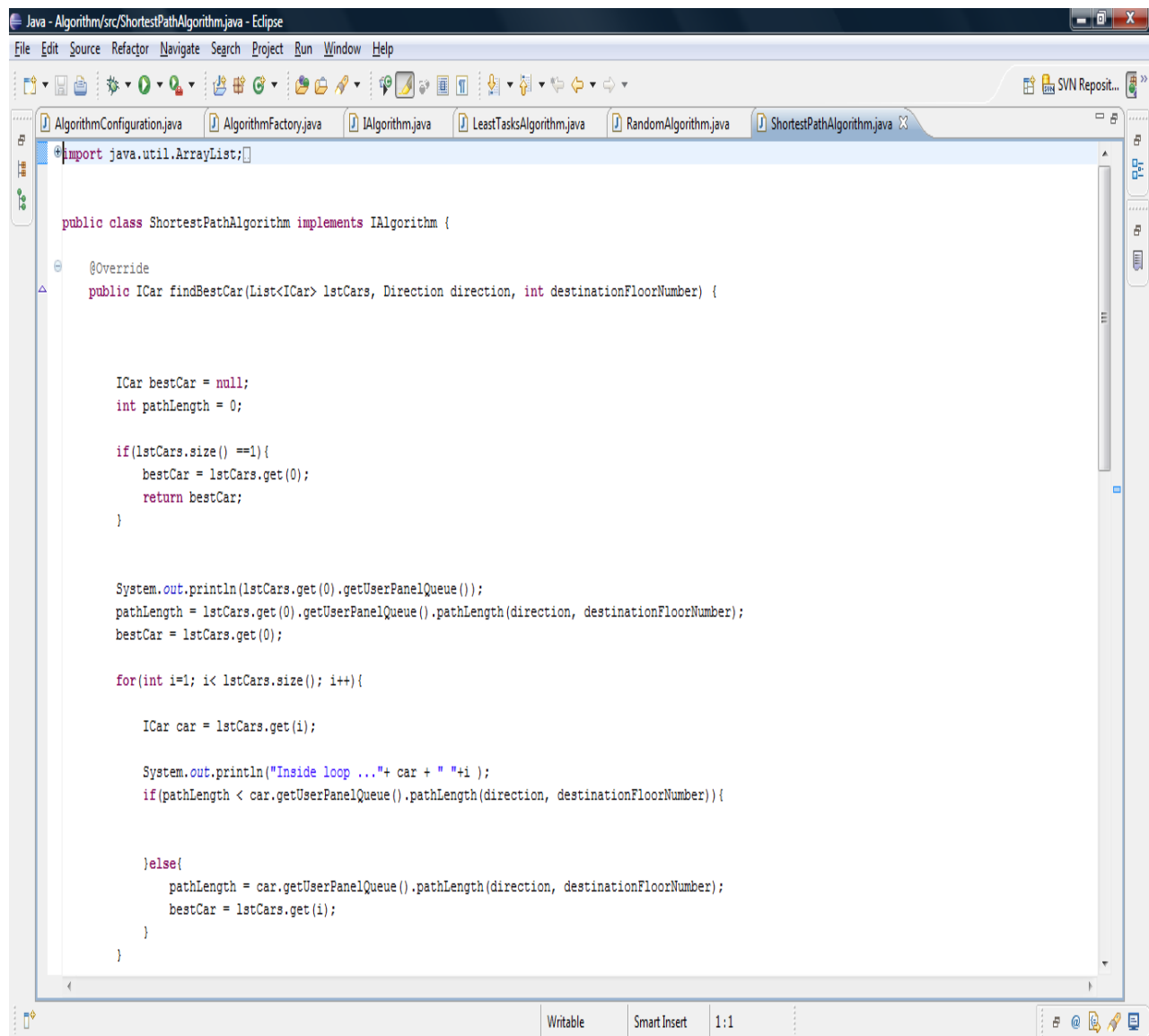
    public static String getDoorConfiguration() {
        return doorConfiguration;
    }

    public static void setDoorConfiguration(String doorConfiguration) {
        GuiConfiguration.doorConfiguration = doorConfiguration;
    }

    public static String getUserPanelColor() {
        return userPanelColor;
    }

    public static void setUserPanelColor(String userPanelColor) {
```

Figure 14. Screen shot of Admin GUI configuration code snippet



```
Java - Algorithm/src/ShortestPathAlgorithm.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help

import java.util.ArrayList;

public class ShortestPathAlgorithm implements IAlgorithm {

    @Override
    public ICar findBestCar(List<ICar> lstCars, Direction direction, int destinationFloorNumber) {

        ICar bestCar = null;
        int pathLength = 0;

        if(lstCars.size() == 1){
            bestCar = lstCars.get(0);
            return bestCar;
        }

        System.out.println(lstCars.get(0).getUserPanelQueue());
        pathLength = lstCars.get(0).getUserPanelQueue().pathLength(direction, destinationFloorNumber);
        bestCar = lstCars.get(0);

        for(int i=1; i< lstCars.size(); i++){

            ICar car = lstCars.get(i);

            System.out.println("Inside loop ..." + car + " "+i );
            if(pathLength < car.getUserPanelQueue().pathLength(direction, destinationFloorNumber)){

            }else{
                pathLength = car.getUserPanelQueue().pathLength(direction, destinationFloorNumber);
                bestCar = lstCars.get(i);
            }
        }
    }
}
```

Figure 15. Screen shot of “ShortestPathAlgorithm” code snippet

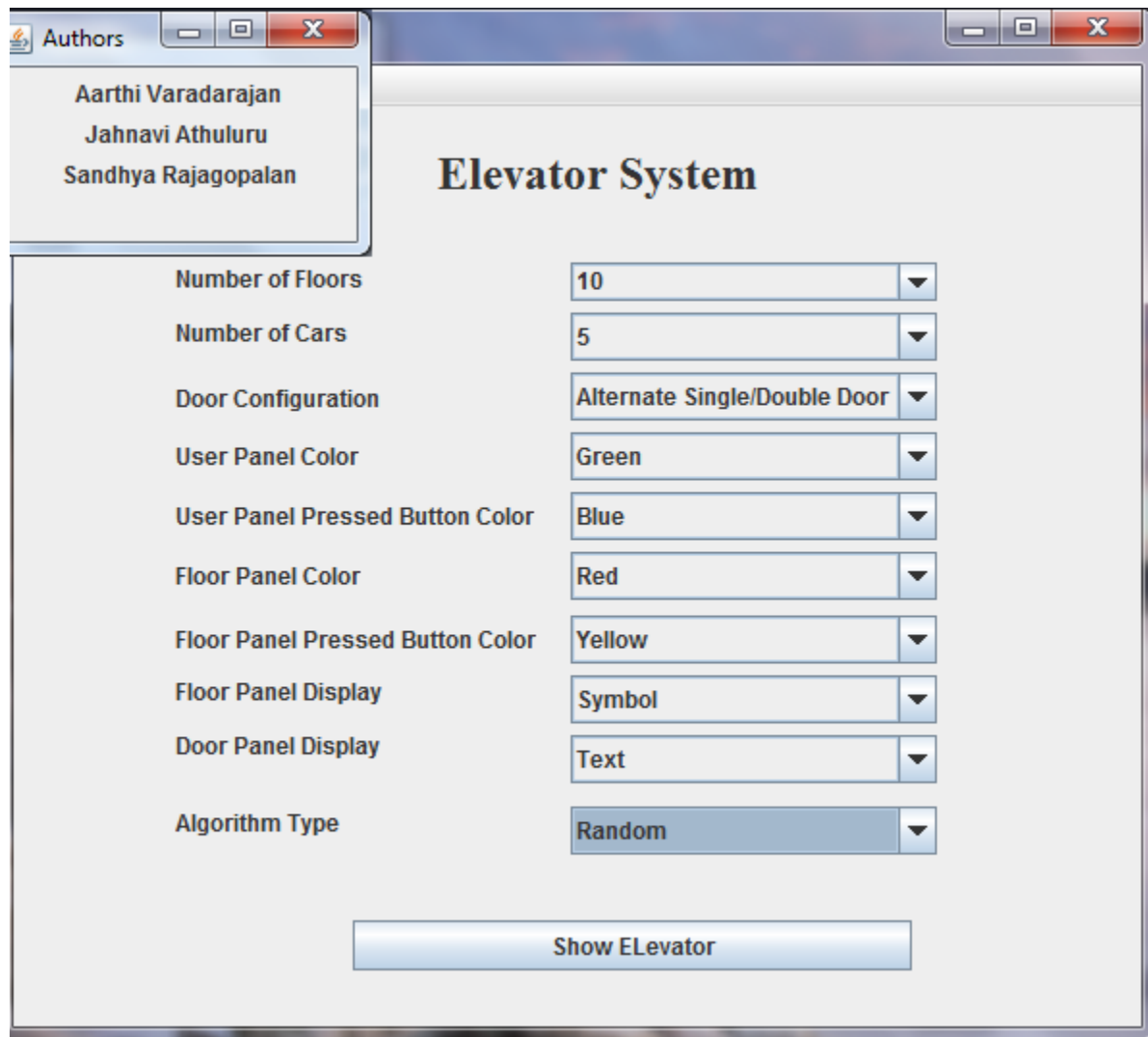


Figure 16. Screen shot showing Admin Console GUI for the elevator system

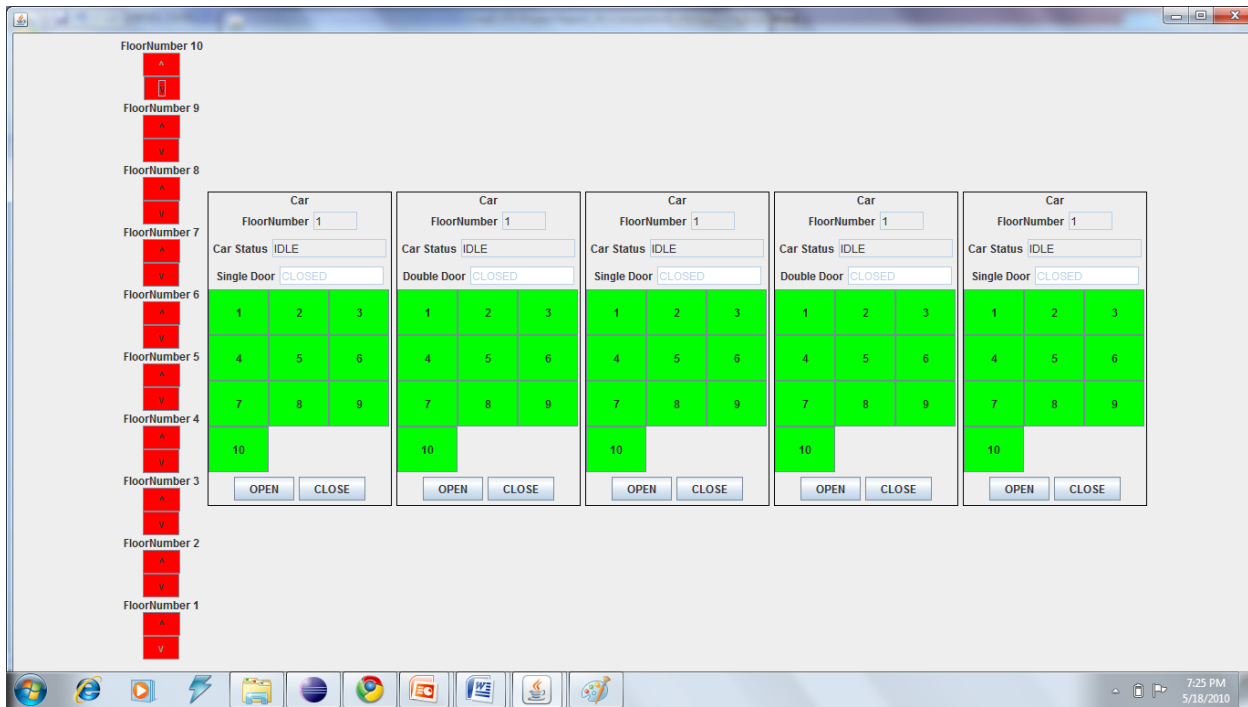


Figure 17. Screen shot showing the elevator system created based on the configuration parameters specified by admin

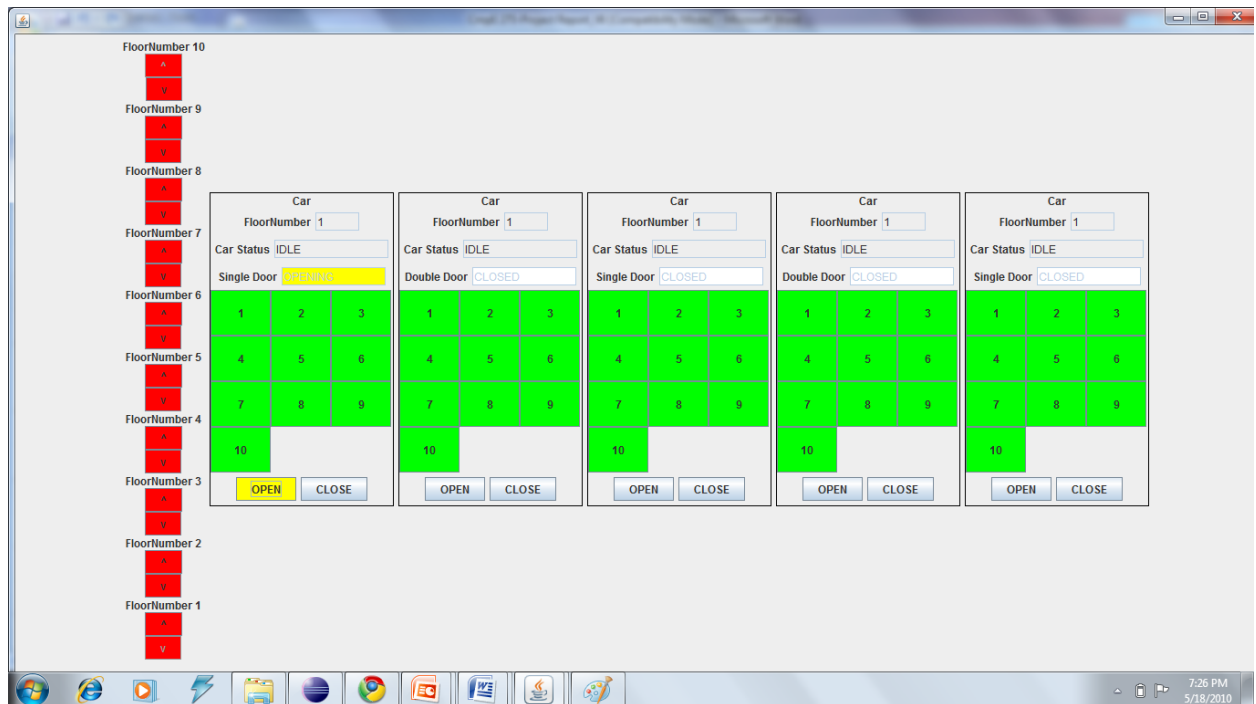


Figure 18. Screen shot showing door “Opening” state for Car 1

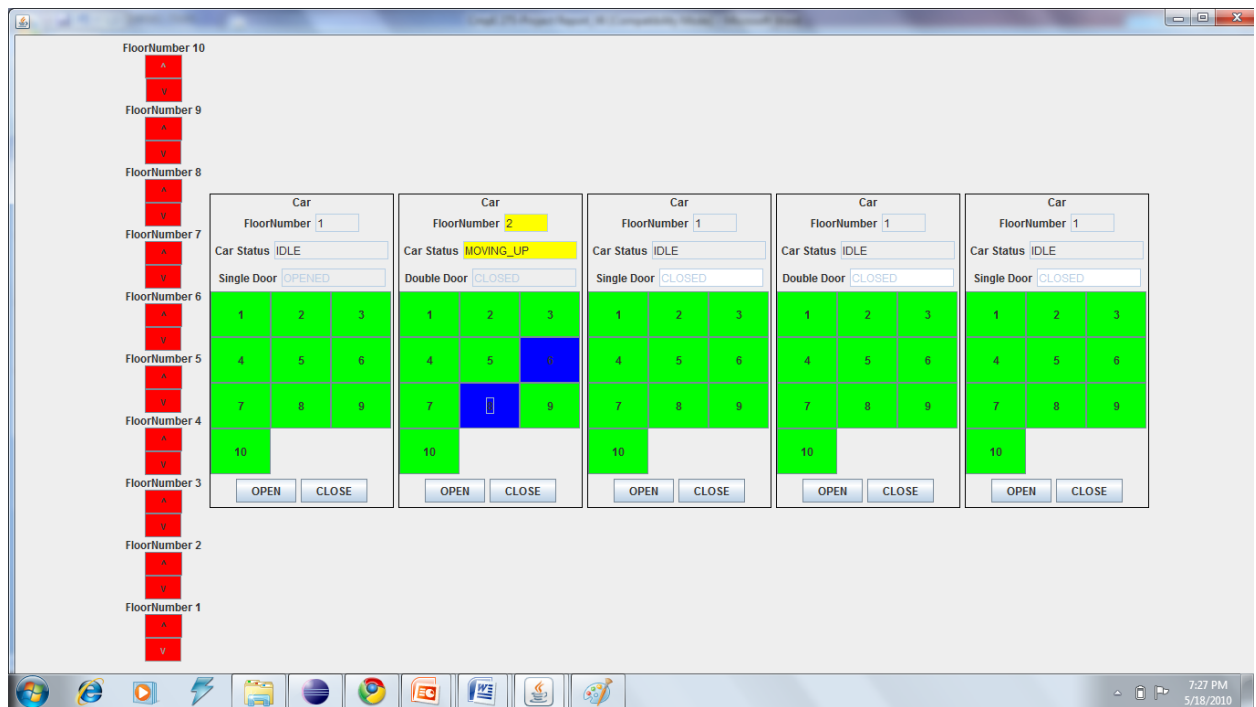


Figure 19. Screen shot showing “Moving up” state for car 2

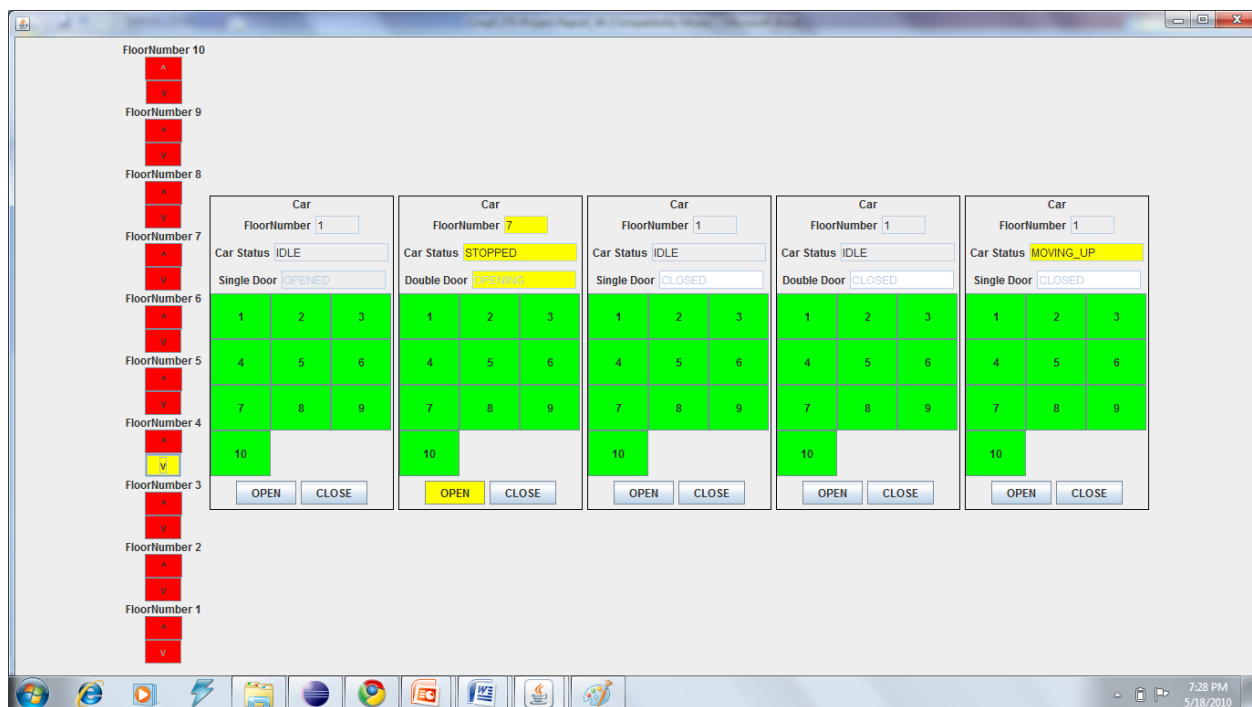


Figure 20. Screen shot showing processing of floor panel request

14. Comparison with existing system

The following table describes the differences between the project that is developed and the already existing project. Some of them are

Features	Existing Elevator System	Current Elevator system
Similarity	Similar to software modules	Software components
Component Deploy ability and Configurability	Cannot be deployed or configured independently. Like software modules.	Can be deployed and configured as a whole component and also independently
Testability	Easy. Testing is done for the software module as a whole	Difficult. Testing done within each component and across components. If reusing the component, reusability tests have to be performed.
Design and Analysis	In terms of functions or sub functions	In terms of reusable components

Table 4: Comparison between existing and current elevator systems

15. Future Directions

In the current design of component based elevator system, though the components are pluggable and replaceable, the GUI for components like user panel and floor panel are closely coupled with the component and are inseparable. In future the design of this elevator system can be modified so that the GUI part is also componentized and made replaceable.

Another future enhancement will be to implement a remote meta controller that remotely controls cars that are running in distributed systems. The solution we have proposed for this feature is to make the UserPanelQueue component a JMS queue running on a JMS provider.

16. Conclusion and Learning

By developing this elevator simulation system as a component-based system, we were able to demonstrate how in real time component development as well as CBS development works. We learnt how components can ease the development of complex systems. These are following observations made about CBD during the course of this project:

- It facilitates distributed development of components
- It allows for easy assembly of components as components are independently well-tested.
- It has drastically reduced the lines of code for each component as each component is highly encapsulated and cohesive.
- It allows us to design usable and stable interfaces.
- It also allows for easy extensibility due to clearly defined interfaces.
- It provides high flexibility to component users as they can easily configure components without changing their application code.

17. Reference

- Component-Based Software Engineering: Technologies, Development Frameworks, and Quality Assurance Schemes.