22c:022 Objected-Oriented Software Development
Fall 2013

Course Project

Due date 1: Friday, Nov 22 by 7pm (Part A)
Due date 2: Friday, Dec 13 by 7pm (Part B)

This is a project to be done in teams of 3-4 people. The grade for the project will be given on an individual basis. Each student will be asked to submit an evaluation on a form provided by the instructor of how well they and their teammates performed as team members. Evaluations are confidential and will be incorporated into the calculation of the project grade.

The goal of this project is to design and implement a system satisfying the requirements listed below.

The project handout is divided in two parts. **Part A** presents the application and the system requirements, and discusses an initial set of document that you will need to submit by the first due date above. **Part B** specifies a subset of the complete set of requirements that you are to implement.

You may be given some additions/changes to the requirements later in the semester.[1] A well-designed system will make accommodating these changes relatively easy.

A Controller and Model for an Elevator System

The software system to be developed is the controller for an elevator system like those typically found in multistory buildings. For simplicity we will consider a building with three floors. Also for simplicity we will also assume that, which each floor has sliding doors into the elevator shaft, the elevator's car itself has no internal doors. The overall system allows people to call the elevator with requests to go up from the first and the second floor or down from the third and second floor. The buttons inside the elevator allows passengers to ask the elevator to go to a specific floor, or to stop. The keypad on the first floor allows a maintenance person to insert a key and put the elevator in maintenance mode. The elevator is also connected to the building's fire alarm system which can put the elevator in emergency mode. See later for details on the expected behavior of the elevator when the maintenance or the emergency mode is switched on.

[1] As observed in class, such changes are common in real software development projects.

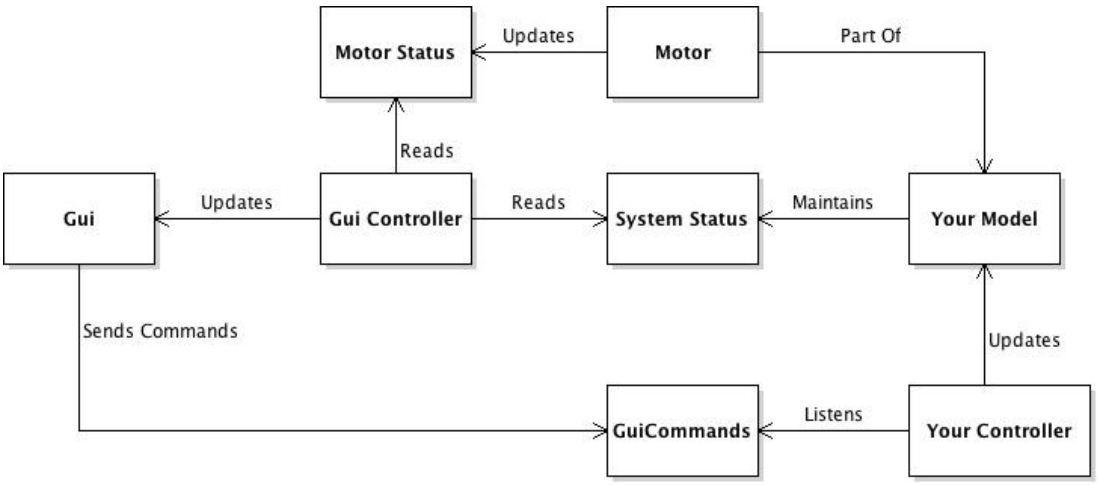

Figure 1: Overview of the Elevator System.

Implementation Set Up

Your Elevator subsystem should be implemented in Scala. We have provided for you a graphical front-end simulating the elevator's physical interface, as well as a module simulating the elevator motor. The files are included in an SBT project contained in the file `ElevatorProject.zip`.

A system diagram of the overall system, with some basic flow of information is shown in Figure 1. As you can see, your subsystem consists of two main components, a controller and a model, in the sense of the Model View Controller paradigm. This components only interact with the Motor, SystemStatus, and GuiCommands components.

Your model component will have to maintain the SystemStatus component, defined as an object in the `SystemStatus.scala` file. SystemStatus is the component that the GUI controller GuiController is watching for changes. GuiController re-renders the GUI view for you as changes are made to SystemStatus.

Your controller module will have react to requests sent to the elevator. In the current simulated setting, the requests will come from the Gui component via the GuiCommands component. The latter component defines a number of requests that the elevator should respond to, such as requests from the buttons on each floor (e.g., to go down from the third floor, one to go up from the second, and so on) and requests from the buttons inside the elevator (e.g., to go to the first floor, to stop and so on). These requests are encoded as methods of the object guiOutput. You should replace the body of these methods to suitable calls to your elevator controller component.

You should not add or change any of the existing code in the provided Scala files except for that in `GuiCommands.scala`.

Your implementation will have the following interactions with the system already implemented.

- It will send commands (and ask for information) to the motor.
- It will change the values of the Boolean fields of the object SystemStatus.

- It will react to method calls issued by `guiOutput` defined in the `GuiCommands.scala`.

Your system should satisfy the following requirements specified below. These requirements are listed in somewhat random order and may not be complete or precise. This is intentional, to reflect the sort of input that a real customer might provide. It will be your job to complete and disambiguate the requirements as needed by relying on your knowledge of elevators and by checking with the TAs who will act as your customers.

Requirements

Let's define a *request* to be either a command for the elevator to go to a certain floor (issued from inside the elevator), or a call for the elevator to go up or down (issued from outside the elevator).

All requests to the system should be eventually be met unless the system is been put in maintenance or alarm mode. It the elevator is idle at a floor, that floor's door should stay open until either a passenger selected a floor from inside, or the elevator is been requested at another location.

All buttons should be lit once pressed. They should remain lit until the corresponding request has been serviced.

The elevator services requests according to the following protocol.[2] If a new request arrives while the elevator is idle, the elevator will move in the direction of the requested floor. Requests that arrive while the elevator is moving are serviced only if they are in the current direction of the elevator, which requests to the closest floor serviced first. Requests in the opposite direction are delayed until the elevator has serviced all of those in the same direction of its movement. At that point, the elevator reversed direction and the whole process repeats.

The doors at a floor should open when the elevator reaches that floor to let off or to pick up passengers. They should not should not open unless the elevator is there.[3] The motor should not get a down command when the elevator is at floor 1, and should not get an up command when the elevator is at floor 3.

The stop button should stop the elevator at its current location—essentially halting all progress. When the stop button is pressed again the elevator should resume where it left off. The stop button should remain lit while the elevator is halted. Note pressing the stop button should should be the only way to indefinitely delay requests.

When the alarm is switched on, the elevator will immediatly drop all queued up request and go to the nearest floor, regardless of the direction was moving to, and open the doors. When the alarm is turned from on to off the elevator will go the first floor and reset. Once at the first floor it will resume regular operations.

When the maintenance switch is turned on, the elevator will stop accepting requests, service all of the pending ones and then go to the first floor, and open the doors, to let the maintenance crew in.

[2] A similar protocol, aptly called the *elevator algorithm*, is used in hard disks to service read and write requests. See `http://en.wikipedia.org/wiki/Elevator_algorithm`.

[3] The GUI will show red doors if a door is opened to an empty shaft, while it will show green doors to doors opened when the elevator is there.

Once the maintenance is complete and the switch has been turned back to off, the system resets itself similarly to how it reset after an alarm is switched off.

Part A

The system must provide the following functionality. The sub-functions marked with (M) can be requested only by a maintenance crew member; those marked by (A) can be requested only by the alarm system. The others are performed by human users, including of course maintenance crew members.

1. Moving Passengers
   (a) Call the elevator to a location.
   (b) Tell the elevator to drop off a passenger at a location.
   (c) Stop elevator at current location (via stop button)
2. Switch maintenance mode on and off (M)
3. Switch emergency mode on and off (A)

Deliverables

You are to produce the following material to document the results of your analysis and design at a sufficient level of detail. These deliverables should be submitted by **Due date 1**.

1. A use case specification document for all the functionalities 1a, 1b, 1c, 2, and 3 above.

   Identify a selection of important use cases (at least three) and describe them in "fully dressed" style, as illustrated in Chapter 6 of the textbook. Write the remaining use cases in casual style.

2. A UML use case diagram modeling all the use cases.
3. A *domain-level* diagram for the whole system, expressed as a UML class diagram.
4. Two UML sequence diagrams each based on one of the use cases you provided in fully dressed style.

Submission Instructions

You can create the requested diagrams with any tool for generating UML 2 diagrams. However, each diagram must be submitted in PDF format. If a diagram is too big to be readable in a single page, break it in two pages, duplicating nodes as necessary in the subdiagrams.

You can write the use case specification document with any editor or word processor. However, that too must be submitted in PDF format.

Submit through ICON the 4 deliverables above as 4 different PDF files.

Grading Notes

The documents you submit for Part A will be graded and returned to you, but you will receive no marks for them, only feedback. You will use this feedback to improve and expand those documents as needed for delivery in Part B. However, a penalty will applied to the final grade of the project for each missing deliverable in Part A.

Note: It is in your best interest to produce complete and detailed enough documents for Part A the first time around. After that, you will have time only for improvements, not for a major analysis and design effort.

Part B

This part describes the subset of the system you are to implement and all the deliverables that you must produce at the end of the project and submit by **Due date 2**.

Deliverables

You are to produce the following material by the **Due date 2**.

1. All the documents and diagrams requested for Part A, revised as needed and according to the instructors' feedback.
2. A *design* class diagram for your system. The level of detail that is sufficient for this diagram will be discussed in class later.
3. A working system, delivered as a self-contained SBT project, including all Scala and SBT sources and any non-standard libraries used.[4]

Submission Instructions

All materials should be submitted through ICON. All the document should be in PDF format as for Part A. The SBT project must be submitted as a zipped file.

[4] So that the instructors can build your system from sources and run it on their machines.