DM519 Concurrent Programming
Spring 2014 Exam Project

By Mads Petersen

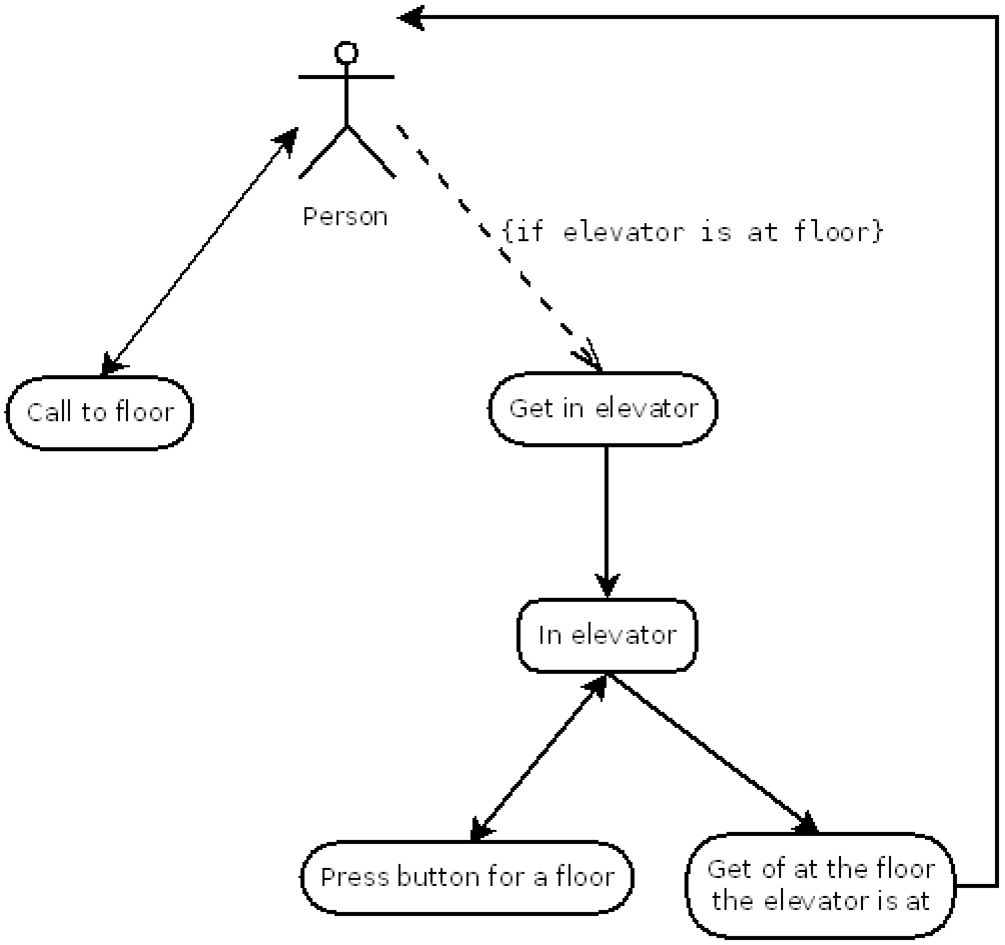
S17

Table of Contents

Modelling

Design decisions

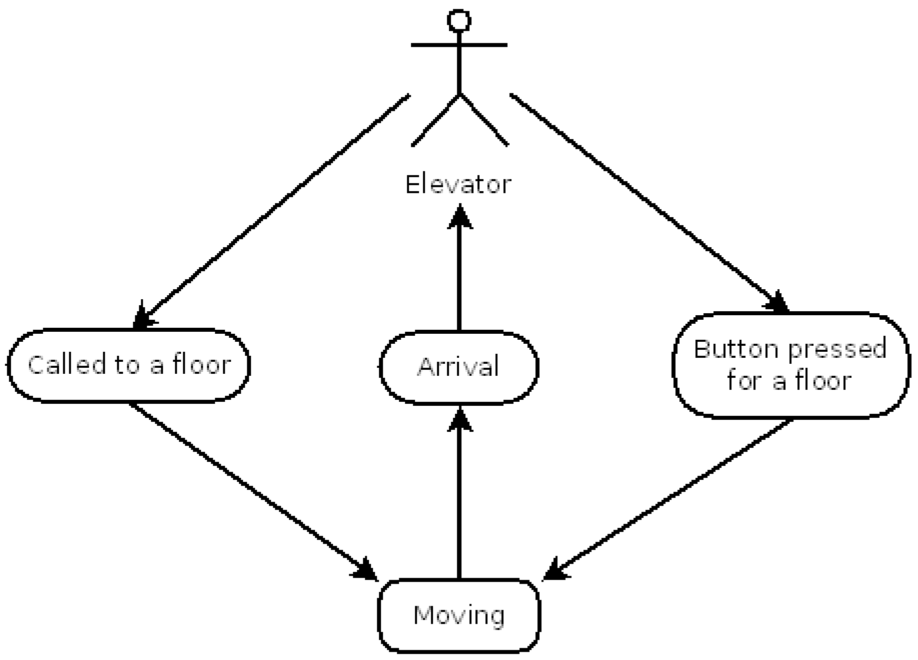
Some specifications have been decided upon for the purposes of this project. As is stated in the assignment, each floor has a call button, and in the elevator there are buttons for each floor and when either of these buttons are pressed all other buttons are deactivated until the elevator reaches its destination. It has been decided that after reaching the destination people have some choices depending on whether they are in the elevator or not. If a person is in the elevator he can press any of the buttons(not taking into account restrictions imposed by clearance levels) or the person can get out of the elevator at the floor it is currently at. If a person is not in the elevator that person can call the elevator to the floor he or she is currently on or that person can get into the elevator if it is at his or her floor. All of these choices are only available when the elevator isn't moving, that is, after someone has either pressed a button in the elevator, or someone has called it to a floor.



Arrows with heads at both ends indicate an action that leads back to the same state, arrows with one head indicates they lead to a new state, dotted lines indicate a conditional action.

This means that a person can call the elevator to a floor without getting in it, and can also press the button for a floor without getting out at that floor.

The elevator is then going to respond either when the button is pressed by someone in the elevator, or when it is called to a floor:



Therefore there are two states for the elevator to be in, either it is moving, or it is not. When it is moving it cannot be called until it arrives. The only influence the different floors have is what restrictions they incur as a result of having a clearance requirement for entering, there are not limit to the amount of people that can be on a floor or any other relevant information about the floors. The limit of people in the elevator at anyone time is going to be 1 for the purposes of the FSP model and then be limited by a variable in the java implementation.

FSP model

The FSP model has a primary process that is the elevator, this has the actions described above, person entering, person leaving, getting called to a floor and a button pressed.
When it is called to a floor or when a button is pressed it goes into the “MOVING” state and buttons cannot be pressed until it arrives. The action “button_pressed”(composed as “elevator.button_pressed) is the action representing someone inside the elevator pressing a button, called is when the elevator is called away to a floor and is followed by the specific call from a floor, compsed as “elevator.called” and “elevator.(floor).call” where the floors are ground, s1, s2 and t.

Person.enters and person.leaves and the action for people entering or leaving the elevator.
The choices:

```
(when(n>0&&n<MAX_LOAD) button_pressed -> BUTTON_PRESS

|when(n>0&&n<MAX_LOAD) person.enters[p:PPL] -> [p].scan -> [p].scanned -> passenger.change[n+1] ->
WORKING

|when(n>0&&n<MAX_LOAD) person.leaves[p:PPL] -> [p].exit.scan -> [p].exit.scanned ->
passenger.change[n-1] -> WORKING


|when(n==MAX_LOAD) button_pressed -> BUTTON_PRESS

|when(n==MAX_LOAD) person.leaves[p:PPL] -> [p].exit.scan -> [p].exit.scanned -> passenger.change[n-1]
-> WORKING


|when(n==0) called -> WAITING

|when(n==0) person.enters[p:PPL] -> [p].scan -> [p].scanned -> passenger.change[n+1] -> WORKING)),
```

Is intended for deciding which actions are available when the elevator is full, not full but not empty, and empty. As it is now the elevator can carry a maximum of 1 person at a time, therefore the options following “when(n>0&&n<MAX_LOAD)” are never available since n is either 0 or 1 and MAX_LOAD is also 1.

The property SECURITY_BREACH, the verifies that someone with a too low security clearance doesn't get off at a floor they shouldn't have access too:

```
property SECURITY_BREACH =   ([id:PPL].return[id][0].enter -> CLEARENCE_0_ENTERS[id]
                             |[id:PPL].return[id][1].enter -> CLEARENCE_1_ENTERS[id]
                             |[id:PPL].return[id][2].enter -> CLEARENCE_2_ENTERS[id]),


CLEARENCE_0_ENTERS[id:PPL] = (person.leaves[id] -> SECURITY_BREACH
                              |button_press[0] -> CLEARENCE_0_ENTERS[id]),


CLEARENCE_1_ENTERS[id:PPL] = (person.leaves[id] -> SECURITY_BREACH
                              |button_press[0] -> CLEARENCE_1_ENTERS[id]
                              |button_press[1] -> CLEARENCE_1_ENTERS[id]
                              |button_press[2] -> CLEARENCE_1_ENTERS[id]),
CLEARENCE_2_ENTERS[id:PPL] = (person.leaves[id] -> SECURITY_BREACH
                              |button_press[0] -> CLEARENCE_2_ENTERS[id]
                              |button_press[1] -> CLEARENCE_2_ENTERS[id]
                              |button_press[2] -> CLEARENCE_2_ENTERS[id]
                              |button_press[3] -> CLEARENCE_2_ENTERS[id]).
```

Does so by continually offering the action of pressing the buttons a person with a given security clearance can press, and only leaving this state when that person gets out of the elevator.
So if a person with clearance 0 where to get in and press the button for floor 1, this would lead to an error state.

The process CAMERA that ensures someone with too low security clearance doesn't go to a floor they are not allowed on simply does so by restricting the actions a person with clearance 0 has to pressing the button for the ground floor, and so forth for higher clearances.

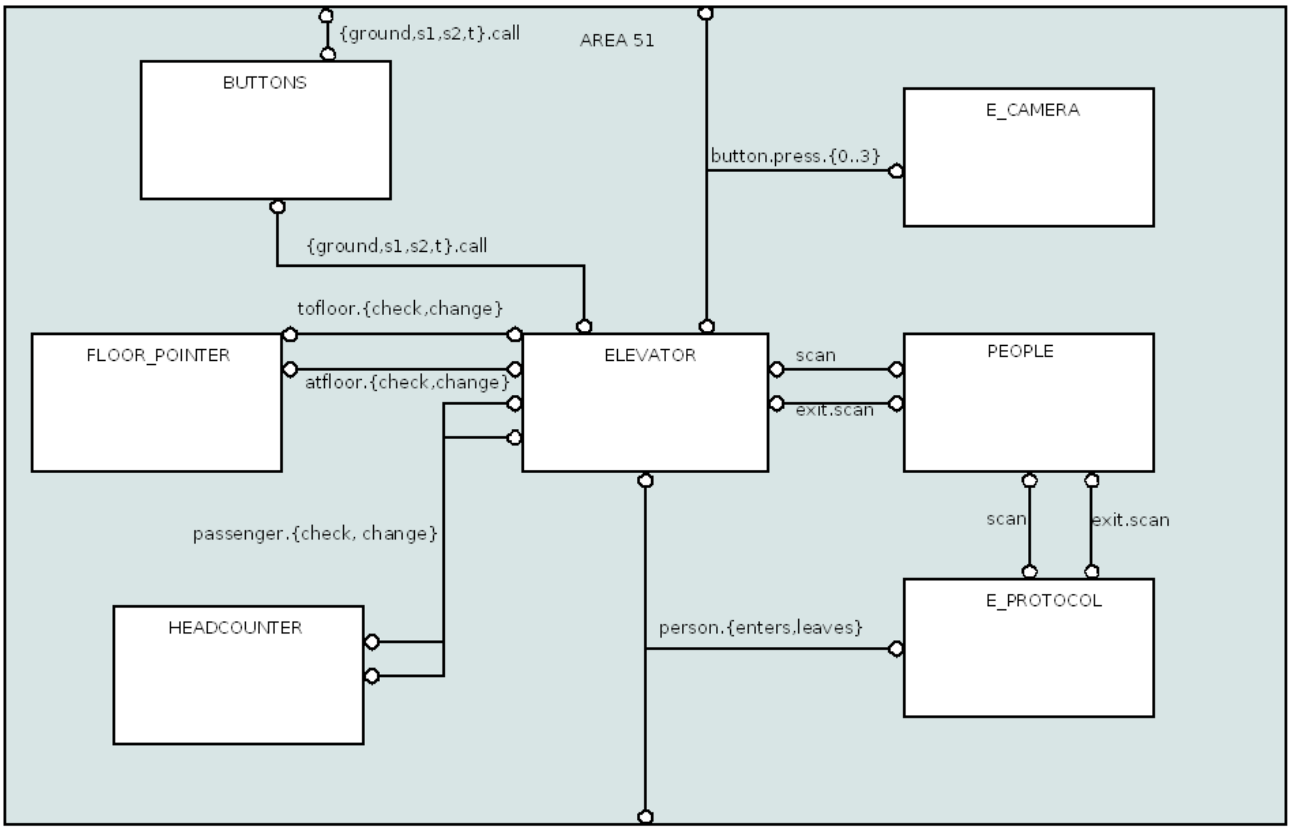
The fact that someone with clearance 0 eventually gets off at the ground floor is verified with with:

```
progress SECURE_EXIT = {elevator.person.leaves[id:PPL]}
```

because person.0 has clearance 0, and it is ensured by CAMERA that this person cannot get off anywhere but at the ground floor, the action elevator.person.leaves[0] can only happen at the ground floor. And therefore verifies that at least 1 person gets off at the ground floor, person.1 can both get off at the ground floor, floor s1 and s2.

The processes FLOORVAR and PPLVAR are used to keep count of how many people are in the elevator(which technically isn't necessary at the moment but it should make it pretty easy to expand to a model that supports more than 1 person at a time) and to keep track of which floor the elevator has been called to and which floor it is currently at. This means that as it is, the elevator moves one floor at a time, then checks whether or not it is at the floor it was called to, if not, then it moves again and so forth. This could be expanded to make it possible to stop while on the way to a specific floor, say for instance the elevator is currently at the ground floor, and is called to the top secret floor and begins moving, then a call comes in(which isn't possible under the current assignment specifications) from floor S1, when the elevator then passes floor S1, which it will since it moves 1 floor at a time, it could check a flag set at floor S1 and then stop, and resume moving to the top secret floor. This would be how most “normal” elevators work, but since the security clearance restrictions would make this difficult, i.e. someone enters the elevator at S1 with clearance less than 2, It is not used here.

Structure diagram



The external actions are person.enters, person.leaves, for people entering and leaving the elevator, button presses while inside the elevator, and calls from the floors. People entering and leaving are regulated by PROTOCOL to ensure that a person has to enter before he leaves, and that it is the same person that is scanned that returns credentials and button presses by CAMERA as it ensures that only those with the appropriate clearance is allowed to press certain buttons.

Analysis

As mentioned in the previous section, safety is ensured with the SECURITY_BREACH property, and liveness with the SECURE_EXIT progress. The absence of deadlocks can be checked as well in the LTSA analyser, and gives the following results:

```
No deadlocks/errors


Progress Check...

-- States: 261 Transitions: 297 Memory used: 6735K

No progress violations detected.

Progress Check in: 1ms.
```

Implementation

Threads vs. monitors

Every part of the system but the people are passive, so these will be represented as threads that can interact with the elevator and the floors. And the elevator will be the monitor with the synchronized methods call and button_press that will be accessed by the threads. And for testing the threads will call the elevator is it represents a person on a floor, or press a button at random if a person is in the elevator. This should then end up simulating how people will interact with the elevator. And if they can move between floors as intended.

Person.java

This class represents people that use the elevator, they have an id and a security clearance, the class implements Runnable so each person can be a separate thread that makes calls to the elevator.

Elevator.java

This is the monitor that represents the elevator, each of the threads calls the synchronized methods in this class.

Floor.java

This is a floor, it can hold any number of persons, it has a security clearance requirement for a person to enter, it has a call method that passes a call to elevator to simulate a call button.

ElevatorGUI.java and GraphicsProgram.java

Together these 2 classes makes up the graphical part of the program. ElevatorGUI.java calls methods in Elevator and GrahicsProgram to get information about how to update the graphics and to actually update them.

Complex.java

This class contains the main method and is only responsible for initializing all the other classes.

Testing

The program should be a simulation of interactions with an elevator, and thus is nothing but a test of the model, there are some problems with simulation though. Firstly, as it is based on randomly calling the elevator and people randomly getting on and off, it is slow to actually get people to leave the ground floor, as everyone starts out on this floor it is the only floor that is called initially and people get in the elevator quickly enough. But as there are still 6 people left on the ground floor when the elevator is full that means it is still very likely that the elevator will get called to the ground floor. And since the elevator can't move beyond the floor to which the person with the lowest security clearance has access it gets more and more unlikely that the elevator gets to a floor the higher up it is and since people doesn't necessarily get off the elevator(as according to the model people can press the button for a floor without getting off the elevator) it means that it takes a few tries before someone gets off at a given floor, combined with the fact that it becomes less and less likely to visit a floor the higher up it is, people end up spread out mostly among the first 2 floors and very rarely get to the upper 2 floors.

But testing shows that the security clearance requirements are implemented correctly, as no-one with clearance below 1 is ever on the first or second floor.

Also there are discrepancies between the amount of people printed on the GUI and the amount of people there should be, sometimes the numbers are going to amount to 11 instead of 10 people(since currently 10 people are hard coded).