Elevator System Design

SYSC 4805

By Group 3:

Abhilash Pravinkumar	100908667
Alex Li	100898386
Peter Aziz	100846141
Jose A Santoyo	100815880

Supervisor: Professor **Dorina C. Petriu**

Department of Systems and Computer Engineering
Faculty of Engineering
Carleton University

April 8, 2016

Table of Contents

Table of Contents

Chapter 1: Introduction

1.1 Problem Statement

The problem concerns the logic required design, implement and test the UML-RT model of an Elevator Control software system that controls a bank of two or more elevators with multiple functionalities. The system requires the elevators to respond to requests from users at various floors and control the motion of the elevator between floors. This is done to develop and simulate real life world applications

1.2 Contribution of each team member

1.2.1 Project Contribution

Contributor	Contribution
Abhilash Pravinkumar	State diagrams, Capsules design, report
Alex Li	Coding, Capsules design, report
Jose	Dependency diagrams, Capsules Design, report
Peter Aziz	Testing, Capsules Design, report

Table 1: Project Contributions

1.2.2Report Contribution

Contributor	Contribution
Abhilash Pravinkumar	2.4 | 3.2 | 3.3 | 5 |
Alex Li	2.2 | 2.3 | 3.2 | 4.5 |
Jose	1 |2.3| 3.1
Peter Aziz	2.1 | 2.2 | 2.3 | 4

Table 2: Report Contributions

Chapter 2:System Requirements & Use cases

2.1 System requirements

For each elevator there are these features:

- A set of elevator buttons used by the users to select the destination floor;
- A set of elevator lamps corresponding to each button, indicating the floors that have been selected as destinations in the current trip;
- An elevator position lamp indicating the current floor of the elevator during the trip;
- An elevator motor controlled by commands to move up, move down and stop. A status sensor indicates when asked by the control system whether the elevator is moving or stopped;
- Emergency breaks that are triggered in different unsafe conditions, as described below;
- An elevator door controlled by commands to open and close the door;
- A door sensor indicating whether there is an obstacle impeding the door from closing.

For each floor there are:

- A floor door for each shaft controlled by commands to open and close the door;

- Up and down floor call buttons, used by a user to request an elevator for going in a certain direction (up or down).
- A corresponding pair of floor lamps indicating the direction(s) already requested.

At each floor and for each elevator there is a pair of direction lamps.For the top and bottom floors there is only one floor call button and corresponding direction lamp. There is also an arrival sensor at each floor in each elevator shaft to detect the arrival of an elevator at the respective floor.

Emergency brakes are applied under these conditions:

- if an elevator is commanded to stop but will not stop at the desired floor;
- if an elevator is commanded to move but will not move;
- if the doors are commanded to open when the car stops at a floor, but the doors will not open;
- if the elevator keeps going after reaching the highest floor on its way up or the lowest floor on its way down.

Once a floor call button is pressed by a user, one of the elevators will be dispatched by a central coordinator to serve the request. After entering the elevator, the passenger presses a selected elevator button to indicate his destination floor. The elevator then moves up/down to the destination floor. To increase the utilization of the entire system, the elevator may stop at other floors on the way to service other requests for the same movement direction. The elevator stops at the nearest requested floors for all the requests assigned to it by the central coordinator. When all calls are served, the elevator stops and waits for the next call to arrive, and the above cycle repeats again. The control policy should comply with a common rule that requests are to be served on a first come-first-serve basis, except for other requests in the same direction that could be served simultaneously. So, when the elevator is heading in a particular direction, the coordinator will assign to it other requests heading in the same direction along the way. After serving all the requests assigned by the coordinator in a direction, the elevator will be assigned requests for the opposite direction. An elevator will move up and down as long as there are outstanding requests to be served.

2.2 Use Case Diagram

The use case diagram of our system is described in Figure 1 below:

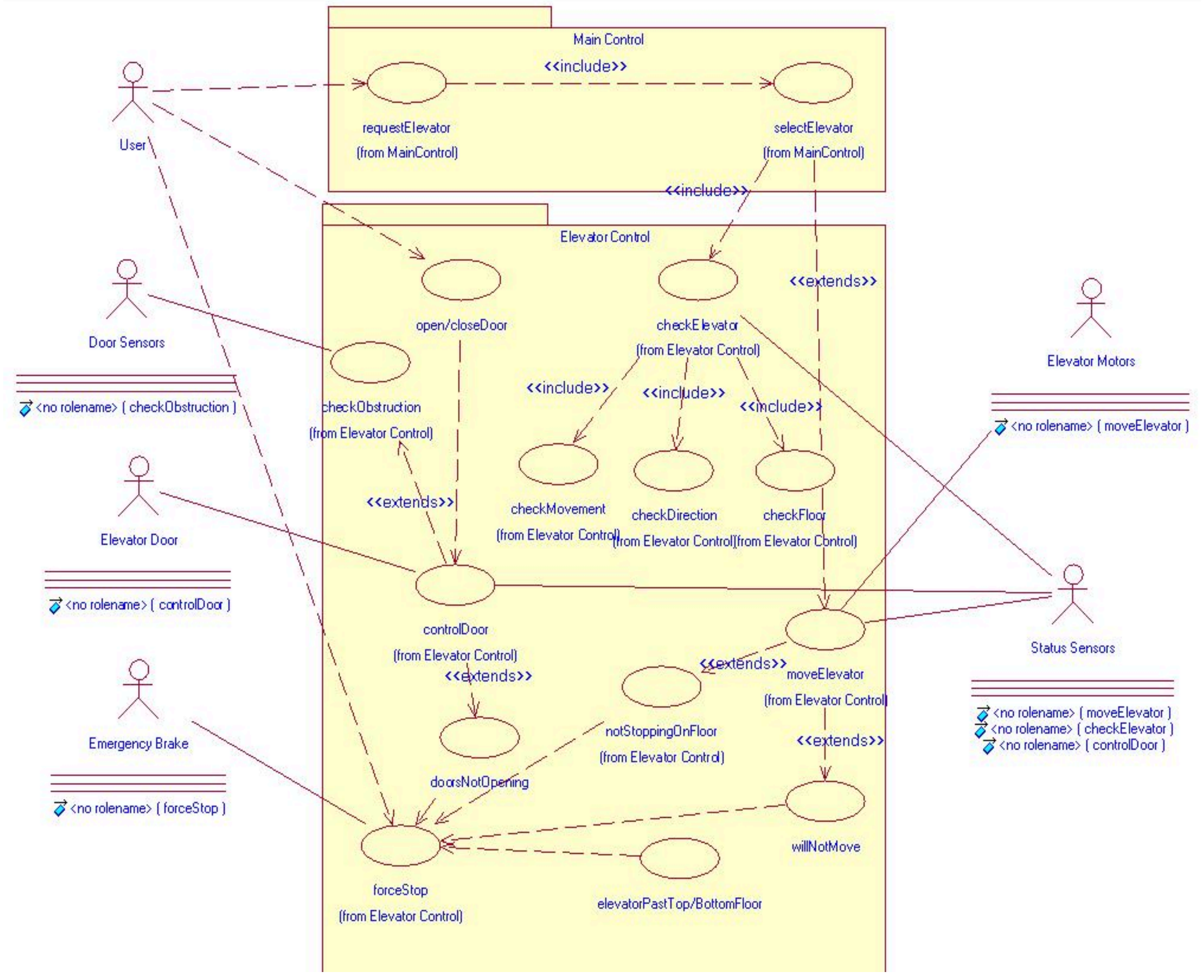

Figure 1: Use Case Diagram

2.3 Use Case Description

Use case name	EmergencyBrakes
Summary	Elevators stops when it does not listen to the uses command
Participating actors	**Primary actor:** Passenger. **Secondary actor:** EmergencyBrake
Flows of events	1. User gets inside the elevator. 2. User selects destination 3. Elevator does not follow the pressed button or user command. 4. Emergency stop is applied or a button is pressed by the user to stop the elevator.
Alternative Flows	The elevator follows the command pressed by the user.
Entry Conditions	User is inside the elevator selecting a floor
Exit Conditions	Brakes applied and user leaves the elevator

Use case name	Read Sensor

Summary	Elevator Controller reads the status of the various sensors to determine the location of the elevator at the current time.
Participating actors	**Primary actor:** Elevator Control
Flows of events	1. Receives message from main controller 2. Polls sensors for the status of the elevator. 3. Sends current status back to the main controller.
Alternative Flows	-
Entry Conditions	Passenger calls elevator
Exit Conditions	Current elevator status is sent to main controller.

Use case name	FloorLobby
Summary	Passenger requests elevator
Participating actors	**Primary actor:** Passenger **Secondary actor:** LocalController

Flows of events	1. Passenger requests elevator 2. Request is sent to MainController 3. MainContoller picks elevator 4. MainController send signal to LocalController to move elevator to destination
Alternative Flows	1. Passenger requests elevator 2. Request is sent to MainController 3. All elevators are busy 4. Users waits for an elevator to be free 5. MainController send signal to LocalController to move elevator to destination
Entry Conditions	Elevator has arrived on requested floor
Exit Conditions	Status is displayed

Use case name	Doors sensor
Summary	DoorSensor reads the status of the sensor to make sure no obstacles are blocking the door

Participating actors	**Primary actor:** ElevatorDoor, Passenger
Flows of events	1. Elevator Arrives at required floor 2. localController opens door of the elevator 3. Elevator waits for 10 seconds 4. Sensors detects nothing is blocking the doors 5. Elevator doors close.
Alternative Flows	3. Passenger presses “Close Door” button. 4. Sensors detects nothing is blocking the doors. 5. Elevator doors close
	3. Passenger presses open door 4. Doors open.
	4. Sensors detect an object blocking the doors 5. Doors open/stay open
Exceptional Flow	1. Elevator is moving 2. Passenger presses “Open Door” 3. Request is not granted.
Entry Conditions	Passenger calls elevator
Exit Conditions	Current elevator status is sent to main controller.

Use case name	SelectFloor
Participating actors	LocalController
Flows of events	1. User selects desired floor 5. Light in the direction where the elevator is moving turns on 6. Elevator moves either up or down depending on the floor selected.
Alternative Flows	1. If the user is at the top floor, user can only go down. 2. If user is at the bottom floor, user can only go up. 3. The elevator does not follow the command pressed by the user. 4. Emergency brakes are applied.
Entry Conditions	Elevator doors open.
Exit Conditions	User gets to the desired floor

Use case name	Display Status
Summary	Displays the status(inside and outside) of the current elevator including

	current floor, if its moving and direction moving
Participating actors	**Primary actor:** Passenger **Secondary Actor:** LocalControl
Flows of events	1. Passenger calls elevator. 2. Elevator controller displays current state of the elevator 3. If elevator moves, the status is changed accordingly
Alternative Flows	-
Entry Conditions	Passenger calls elevator
Exit Conditions	Elevator arrives at destination.

2.4 Changes from Milestone 1

The project evolved over the course from Milestone 1 to Milestone 2 as our team started to break it down into smaller units and understanding the depth and scope of each problem:

Additional Use Cases

More use cases were conceived and added to the use case diagrams with added associations and actors after Milestone 1. Figure 1 below shows the use case diagram during Milestone 1.

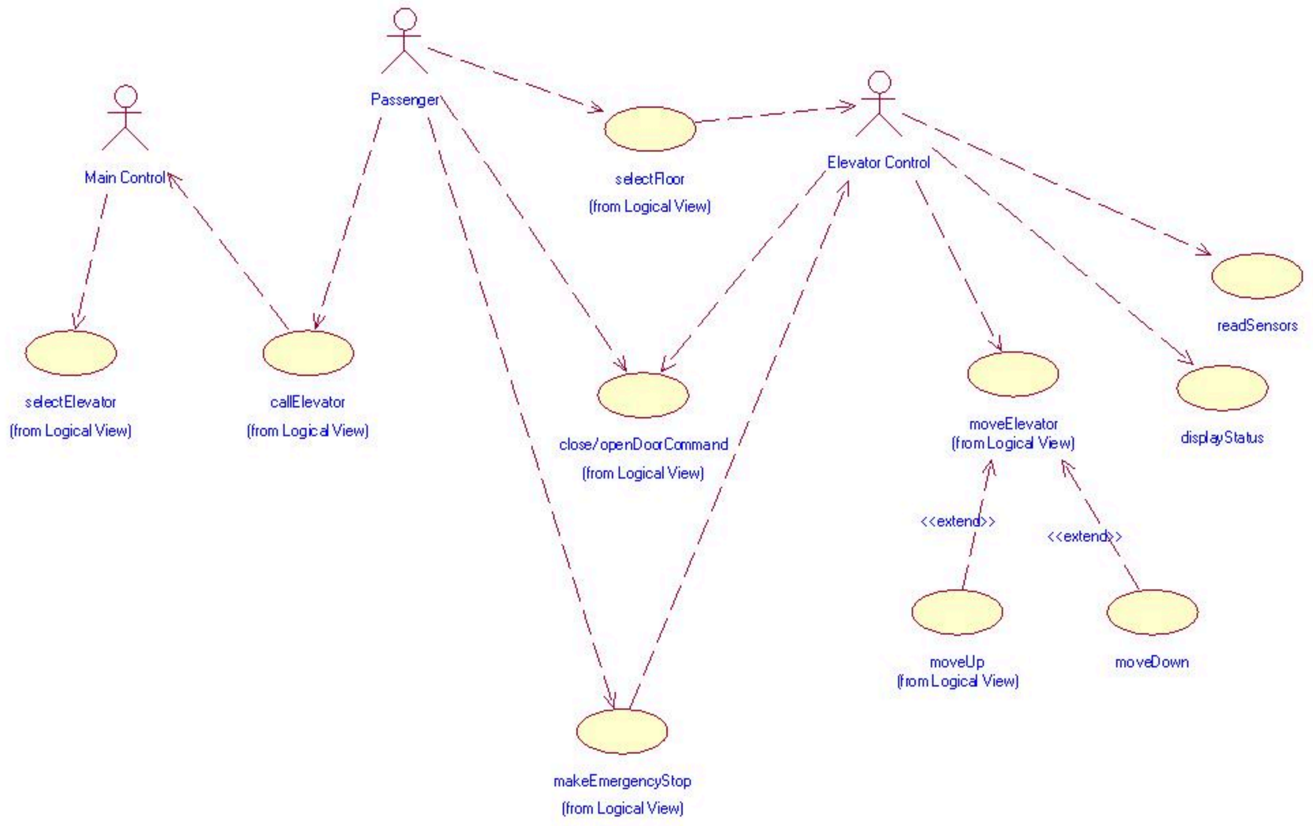

Figure 2: Use case diagram for Milestone 1

The use case diagram presented for Milestone 1 was very simplistic and missed distinction between the various components of the system involved. It also did not have any specifications as to how the sensors are going to be read or how the emergency stop is going to operate. It was also missing certain actors like elevator sensors and emergency brake. The need for these additions was realized by Milestone 2 and added on to the use case diagram presented in Figure 1.

The use case diagram from Figure 1 was used as a template to build the class and capsule diagrams and some of the actor names and components names changed over time.

Chapter 3: Analysis & Design

3.1 Architecture Design and implementation

The designed elevator system comprised of a main controller and a local controller. There is one local controller for each elevator and a top module (i.e. Main controller). The top module manages which elevator gets selected upon user request. The local controller of each elevator instance takes care of all the functional requirements of the elevator system (see section 2.1). The system architecture designed to fully achieve functionality mentioned above is shown in Fig 3 . **MainController** has an **elevator** attribute object type **elevatorInfo** that carries the information of the elevator instance. The way the information is accessed is through the methods that are implemented by the object **elevatorInfo (**see Fig 4**);** setDfloor() sets the destination floor (Dfloor) of the elevator when user requests it at a floor lobby, as well as when user selects floor from inside the elevator; setCfloor() sets the current floor (Cfloor) at which the elevator appears at a given moment; setFloor() sets the floor in the array of floors(i.e. Flags the selected floor); setMoving() sets variable isMoving to 1 when elevator is moving; setUp() and setDown() set the direction of the elevator; setidel() sets the direction of the elevator to idle, meaning that it is not moving; setbreak() sets the emergency break; get methods are used to access to the variable of the **elevatorInfo** object. The attribute **floors** in **MainController** has the array of floors in which the elevators operate. Attribute **requestFloor** stores the value of the floor requested by the user at a floor lobby. The cardinalities are stored in **num_elev** and **num_floor** for the number of elevators and number of floor that the system will support. There is also an **elevators** variable type elevatorInfo array that stores the info object in an array.

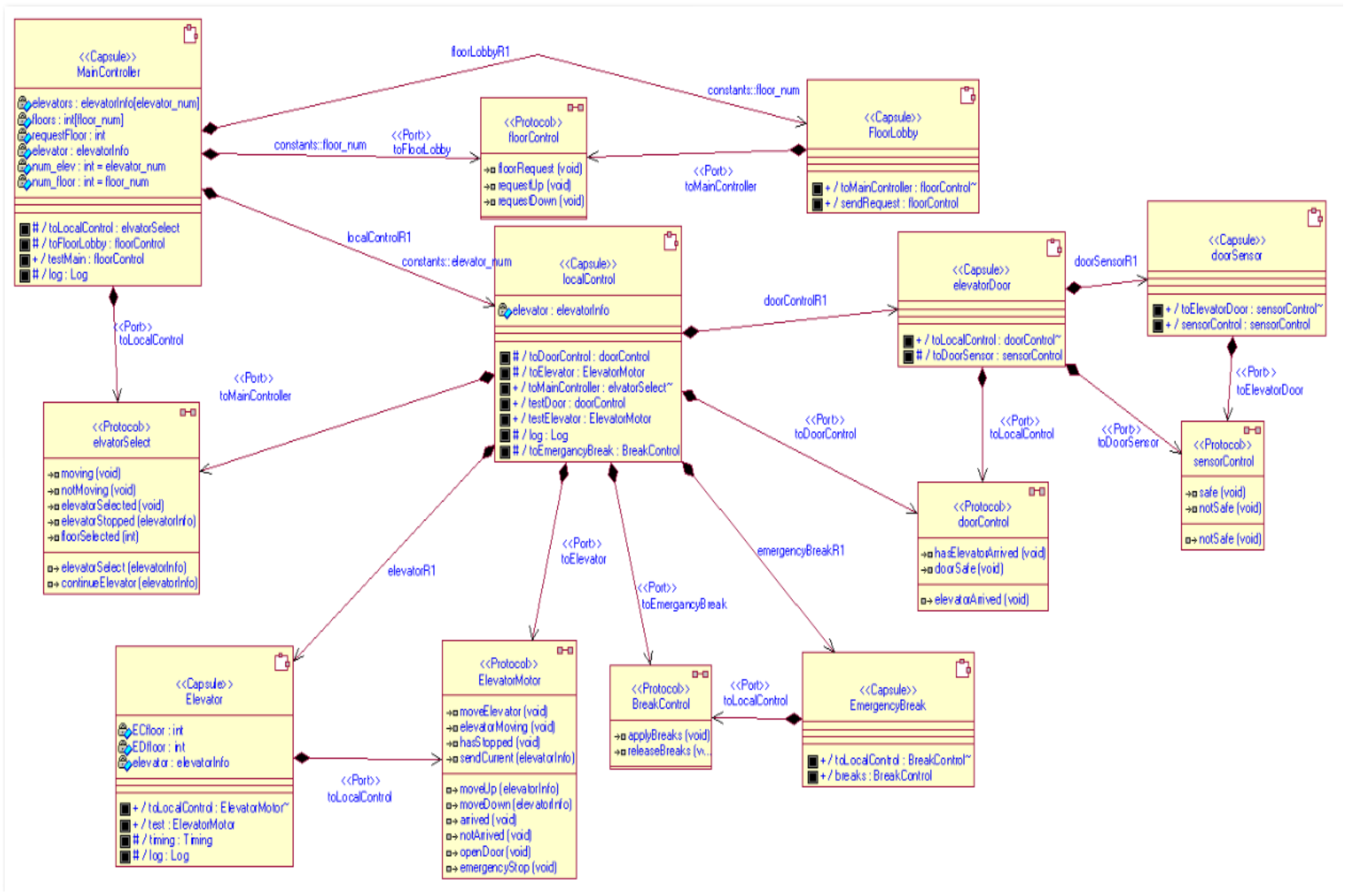

Figure 3. Class Diagram of Elevator System

LocalControl capsule shown in Fig. 3 has as attribute **elevator** type **elavatorInfo** which is shared with **MainController** as previously explained. Each **LocalControl** has its own **elevator** object information and the **MainController** can distinguish one another through the cardinality **index** when communicating through ports and protocols. This last feature is provided in UML Rational Rose. When a message is send from one capsule to another the receiving capsule can use **sapIndex** function to know which instance and from which port was this message sent. Essentially this is how **MainController** knows after analyzing the info from both instances of elevators, which one to select, how to distinguish where the info comes from and to communicate with the desired elevator instance.

The communication protocol between **MainController** and **LocalControl** is **elevatorSelect.** The signals passing through the ports are moving, notMoving, elevetorSelected, elevatorStopped, floorSelected, elevatorSelect, and continueElevator (see Fig. 3). The signals are the events triggers for the states of these two capsules, explained in more detail in section 3.2.2. Apart from communicating with **MainController, LocalController** it also communicates with capsules such as **Elevator**, **EmergencyBreak**, and **elevatorDoor**, shown in Fig.3. **Elevator** capsule contains the number of **ECfloor**, the current floor it is at; **EDfloor**, destination floor; and **elevator** object type **elevatorInfo.** This last one is shared among **LocalControl** and **MainController** as is shown in Fig.4. The communication protocol between Elevator and LocalControl is **ElevatorMotor.** The signals passed are **moveElevator, elevatorMoving, hasStopped, sendCurrent, moveUP, moveDown, arrived, notArrived, openDoor,** and **emergencyStop.** These signals determine the states of the elevator. **EmergencyBreak** capsule communicates with **LocalControl** capsule through **BreakControl** protocol. The event triggers are **applyBreaks** and **releaseBreak.** Lastly, **elevatorDoor** shares information with **LocalControl** using **doorControl** protocol; and the trigger signals are **hasElevatorArrived, doorSafe,** and **elevatorArrived.**

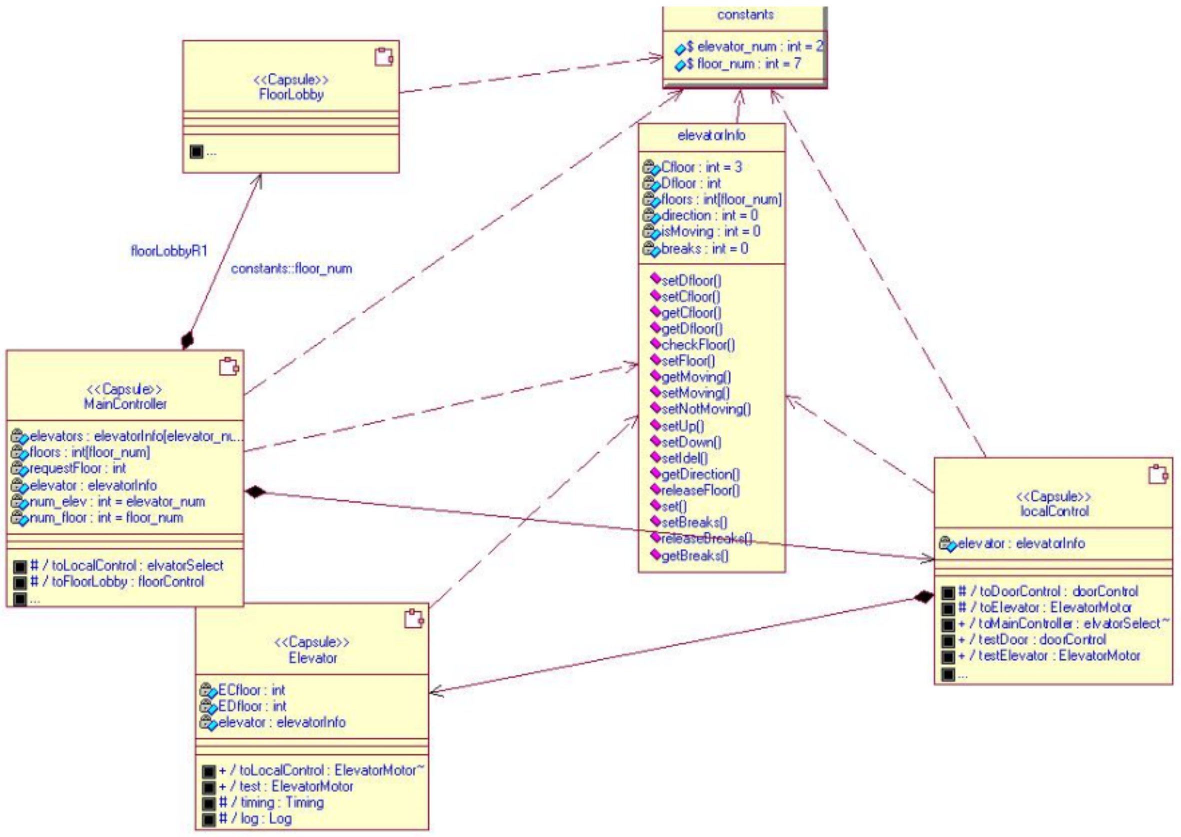

Figure 4. Dependency Diagram of Elevator System

The capsule **doorSensor** reads the sensor and passes that info to **LocalControl** through the **sensorControl** protocol. The signals sent are **safe** and **notSafe.**

When the user request an elevator class **floorLobby** communicates with **MainController** to schedule an elevator for service. The protocol used between these two classes is **floorControl** and the signals passed are **floorRequest, requestUp,** and **requestDown**.

Ports are used in the design and implementation to receive and send signals. The capsules previously mentioned need ports for communication. Testing is also done with ports and test probes, this is explained in detail in chapter 4. Ports are designed in UML-RT to follow a few rules. For example two ports from different capsules have to be conjugate one another in order to be able to establish a connection; ports can be public, conjugated, protected; protected ports are back-end ports. In the design of the elevator system those considerations were taken in order to attain its correct functioning. The system was designed modularly, and tested modularly as the milestones were being completed.

3.2 Description of system behaviour

The system contains multiple capsules that behave independently while connected to the Main Controller and Local Controller as shown in figure 3 above. The Main Controller will handle the request that come from the floors and Local Controller will handle the functionality of the elevator.

3.2.1 Sequence

Figure 5 below shows a sequence diagram on a basic elevator function when a floor requests an elevator.

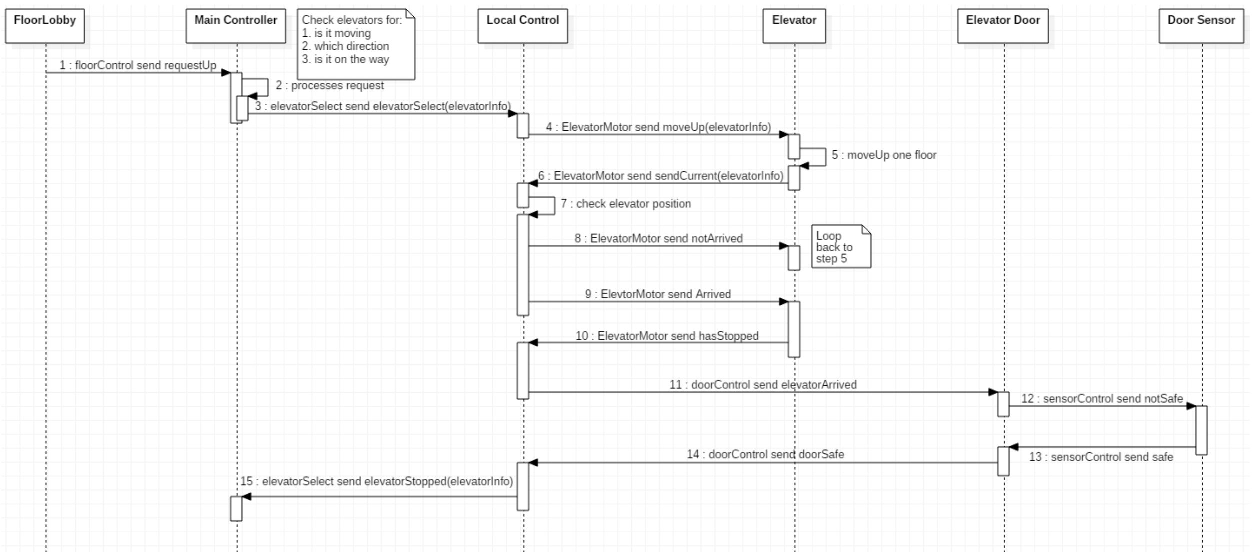
FloorLobby
Main Controller
Check elevators for:
1. is it moving
2. which direction
3. is it on the way
Local Control
Elevator
Elevator Door
Door Sensor
1 : floorControl send requestUp
2 : processes request
3 : elevatorSelect send elevatorSelect(elevatorInfo)
4 : ElevatorMotor send moveUp(elevatorInfo)
5 : moveUp one floor
6 : ElevatorMotor send sendCurrent(elevatorInfo)
7 : check elevator position
8 : ElevatorMotor send notArrived
Loop back to step 5
9 : ElevtorMotor send Arrived
10 : ElevatorMotor send hasStopped
11 : doorControl send elevatorArrived
12 : sensorControl send notSafe
13 : sensorControl send safe
14 : doorControl send doorSafe
15 : elevatorSelect send elevatorStopped(elevatorInfo)

Figure 5: Sequence Diagram handling a request up from floor

The sequence begins when the Main Controller receives a request from the Floor Lobby. The floor lobby can choose to request up or request down signal. This is directly connected to the floorControl protocol, which is connected to the Main Controller so all the signals will go to the Main Controller. An example of an independent capsule is the Floor Lobby capsule.It may send as many signals as it wants while the system is still running. Once Main Controller receives the requests, it will check the elevators for three conditions: is the elevator moving, which direction the elevator is moving, and is the request on the way. Depending on the conditions, it will choose the appropriate elevator. Once the elevator is chosen, Main Controller will send a signal to Local Controller. Local Controller will decide how the elevator will function per request. In this case Local Controller will send a signal to elevator capsule and there it will change the position of the elevator. Once it has met its destination, it will request to open the door through Elevator Door and Door sensor. After the sequence is complete, Local Controller will send a signal to Main Controller saying it has stopped.

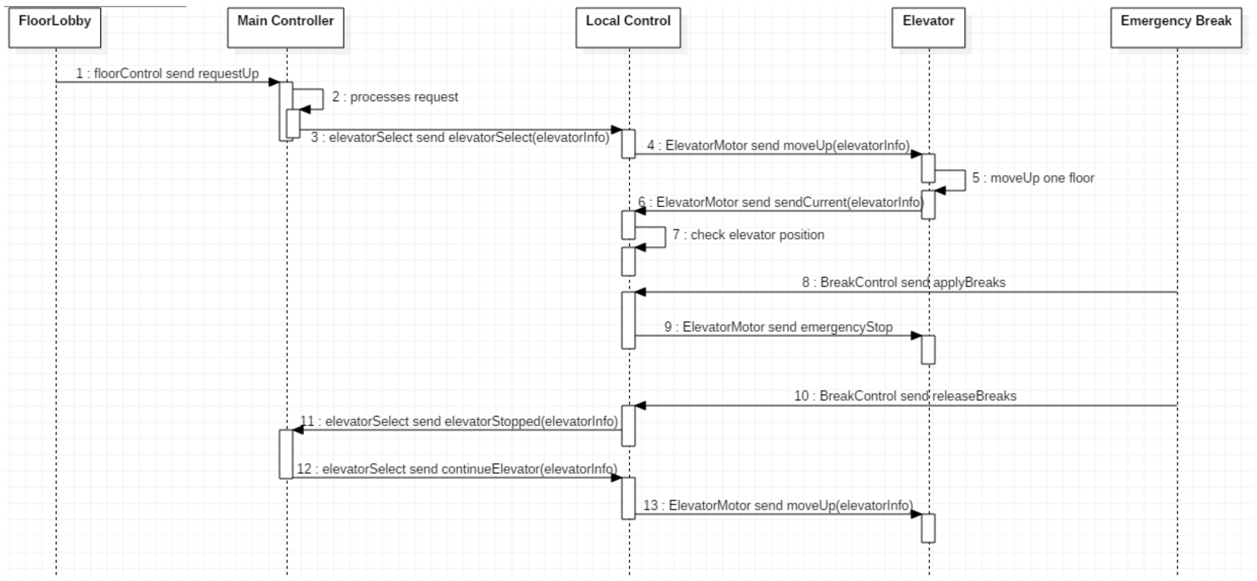
FloorLobby
Main Controller
Local Control
Elevator
Emergency Break
1 : floorControl send requestUp
2 : processes request
3 : elevatorSelect send elevatorSelect(elevatorInfo)
4 : ElevatorMotor send moveUp(elevatorInfo)
5 : moveUp one floor
6 : ElevatorMotor send sendCurrent(elevatorInfo)
7 : check elevator position
8 : BreakControl send applyBreaks
9 : ElevatorMotor send emergencyStop
10 : BreakControl send releaseBreaks
11 : elevatorSelect send elevatorStopped(elevatorInfo)
12 : elevatorSelect send continueElevator(elevatorInfo)
13 : ElevatorMotor send moveUp(elevatorInfo)

Figure 6: Sequence Diagram for Emergency Breaks

Aside from the main sequence that moves the elevator, the system functions a little differently when the emergency brakes are triggered. As the elevator is moving, if the emergency brake is triggered, it puts local controller in a wait state until the brakes are released. Emergency Brake starts with sending a signal applyBreak to Local Controller indicating that the breaks are set. Local Controller must then communicate to Elevator that the brakes are on and must stop. Once the brakes are released the Local Controller tells Main Controller that it has stopped, Main Controller will then tell Local Controller how to continue its route.

3.2.2 State Machines

<u>MainController</u>

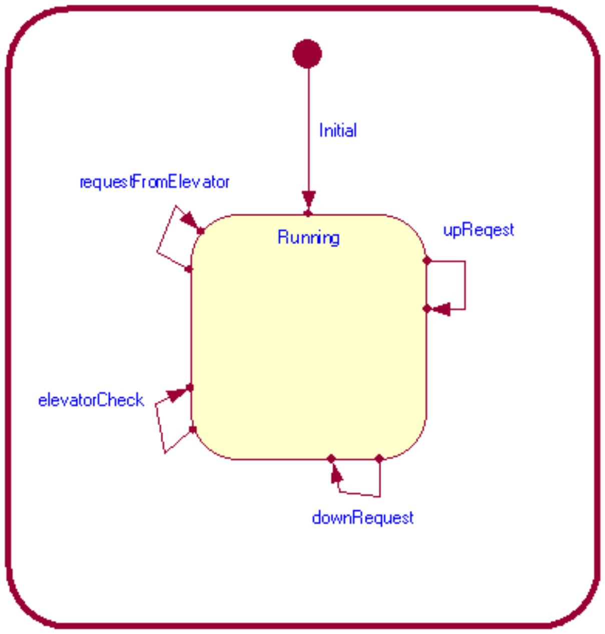

Figure 7: State Diagram for Main Controller

The main controller is responsible for the scheduling and selection of the elevators in the elevator system. It consists of one state called **Running**, which is its default state after initialization. The transition called **upRequest** is triggered by the signal requestUp from the

FloorLobby capsule. When the state machine enters this transition, an action code is triggered which proceeds to select an elevator depending on its proximity to the floor from which the signal was sent and also the direction in which the elevator is headed. It also adds the destination floor to the selected elevator's list of stops so that it knows where to stop. The next transition is called **downRequest**, which also functions in a similar fashion as **upRequest**, but for the opposite direction. It is triggered by the requestDown signal from the FloorLobby. **elevatorCheck** is the next transition which is triggered by the signal elevatorStopped from the LocalControl capsule. If an elevator stops at a floor, it checks to see if this is the current destination floor(destination floor is regarded as the highest or the lowest floor currently requested). If the current floor is not the destination floor, it logs an entry stating that the elevator is headed towards the destination floor. If the current floor is the destination floor, it logs an entry stating that it has stopped at the destination floor. The next transition if the **requestFromElevator**, which is triggered at the floorSelected signal is received from the LocalControl capsule. The action code in this transition adds the selected floor into the list of destinations for the elevator.

<u>**localControl**</u>

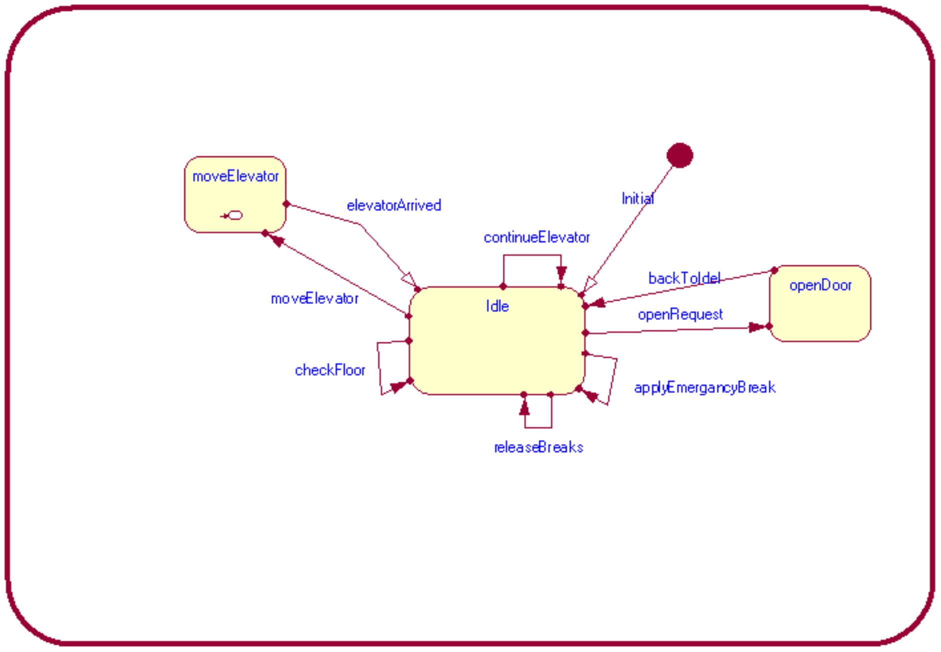

Figure 8: State Diagram for localControl

The default state after initialization for this capsule is called **idle**. The primary transition which is responsible to move the elevator in a specified direction is called **moveElevator**. This occurs when the main controller selects the current elevator and sends it a signal called elevatorSelect.This transition moves the system to the state called **moveElevator**. The entry code of this state selects which direction to move the elevator, according to the floor request. The transition called **elevatorArrived** moves the elevator out of the **moveElevator** state back into the **Idle** state. The next state transition is called **continueElevator,** which is triggered at a signal from the MainController called continueElevator. When this transition is triggered the action code receives data from the main controller indicating that this particular elevator recieved another request, while the elevator is already selected. It is then returned to the idle state. **openRequest** is a transition that is triggered at the signal hasStopped from the **Elevator** capsule. This then sends a signal **elevatorArrived** to DoorControl. This transition leads to another state called **openDoor.** The transition backToIdel is triggered with the signal doorSafe from

DoorControl, indicating that there are no obstructions blocking the door and can be safely closed. The action code for this transition informs the MainControl that the elevator is back in idle position via the signal elevatorStopped. The next transition is called **applyEmergencyBreak** which sends a signal setBreaks() to the elevator, indicating that emergency brakes are to be applied. This transition is triggered by the EmergencyBreak capsule via the signal applyBreaks. The next transition called **releaseBreaks** is complimentary to **applyEmergencyBreak** and releases the emergency brakes from the elevator.

Elevator

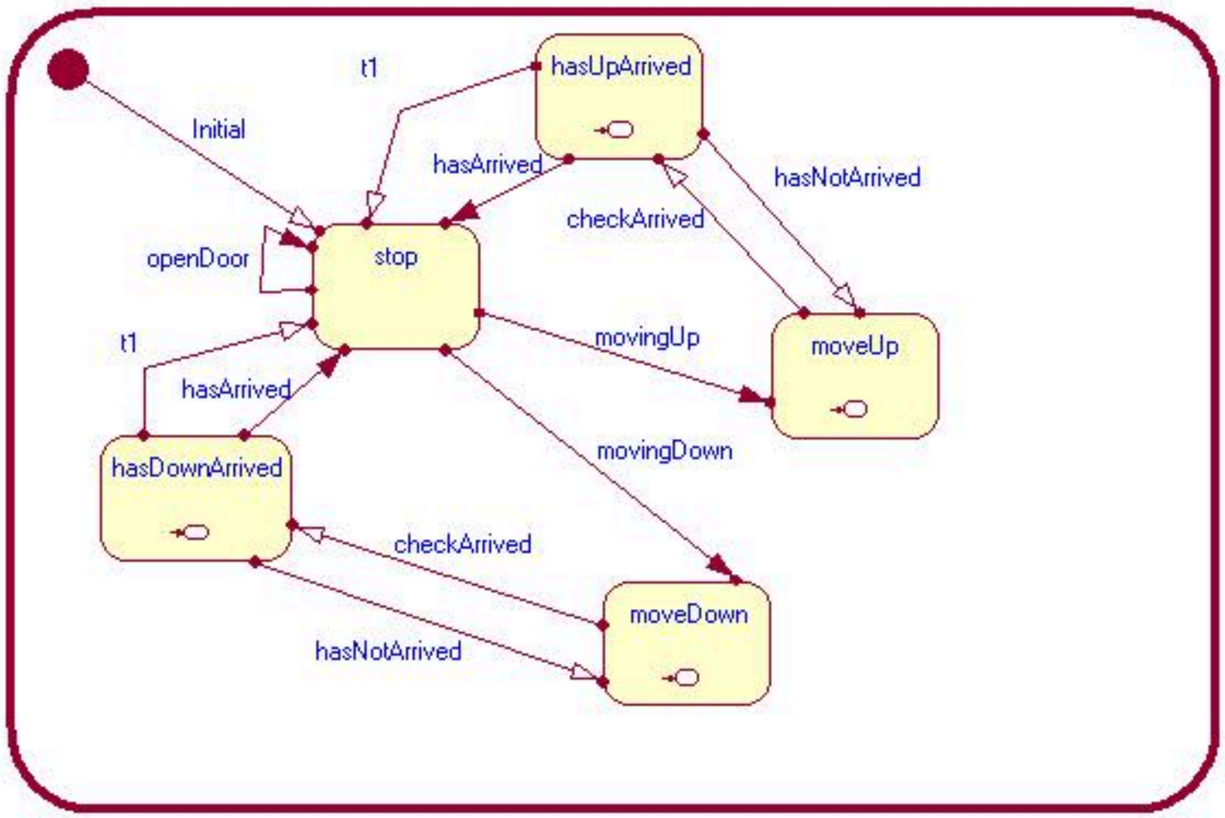

Figure 9: State Diagram for Elevator

The machine enters the **stop** state after initialization. Depending on whether it receives the **moveUp** or **moveDown** signal from LocalControl, the capsule enters the **movingUp** or

movingDown transitions, which lead them to the **moveUp** or **moveDown** states respectively. Both these states are complimentary and have very similar functions, action codes and triggers. This section would go into detail into the **movingUp** transition/**moveUp** state and assume the reader can deduce the **movingDown/moveDown** transition/state themselves. The elevator enters the **movingUp** transition when it recieves the **moveUp** signal from the LocalControl. Once it completes this transition, it arrives into the **moveUp** state where the entry action code constantly keeps updating the current floor every time this state is entered. There is also a timeout function, after which the transition **checkArrived** is triggered. This leads the system on to the state **hasUpArrived**, which sends a signal to the LocalControl capsule to check if the elevator has arrived at the specified floor. If it has, it reverts back to the **stop** state, if not, it reverts back into the **moveUp** state, essentially forming a loop till the floor is reached.

EmergencyBreak

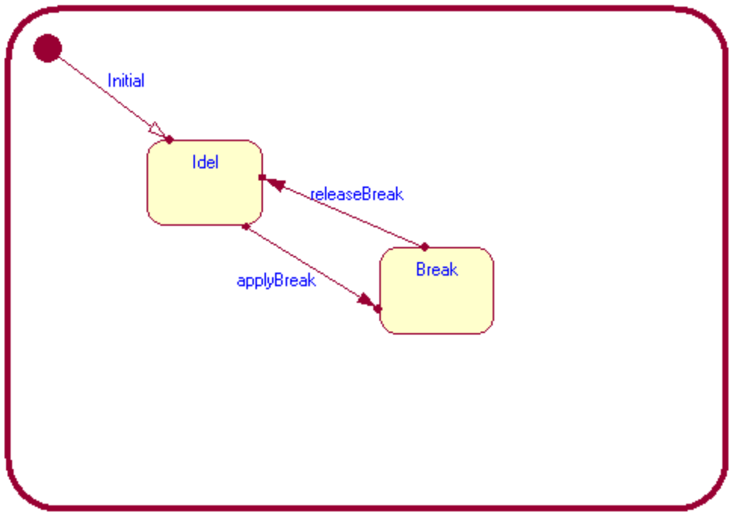

Figure 10: State Diagram for EmergencyBreak

This capsule consists of two states: **Idel** and **Break**. Idel is the default state after initialization. The **applyBreak** transition is triggered by an external action asking the system to apply emergency brakes. This state is called the **Break** state. The **releaseBreak** transition respectively releases the emergency brakes and puts the capsule back in the **idel** state.

doorSensor

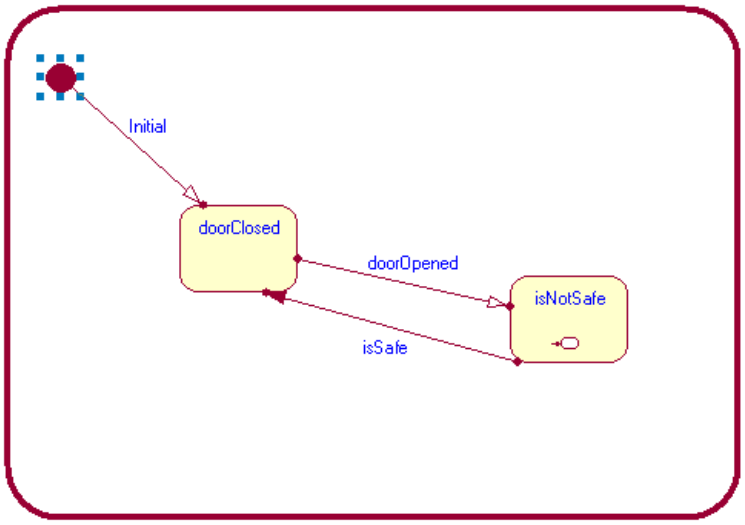

Figure 11: State Diagram for doorSensor

This capsule consists of the default state **doorClosed** and the **isNotSafe**. There is a transition called **doorOpened** from the **doorClosed** to **isNotSafe** which is triggered by the LocalControl when the door is safe to be opened. There is another transition called **isSafe** from **isNotSafe** to **doorClosed** which is triggered from the LocalControl when the door is actually safe to close

elevatorDoor

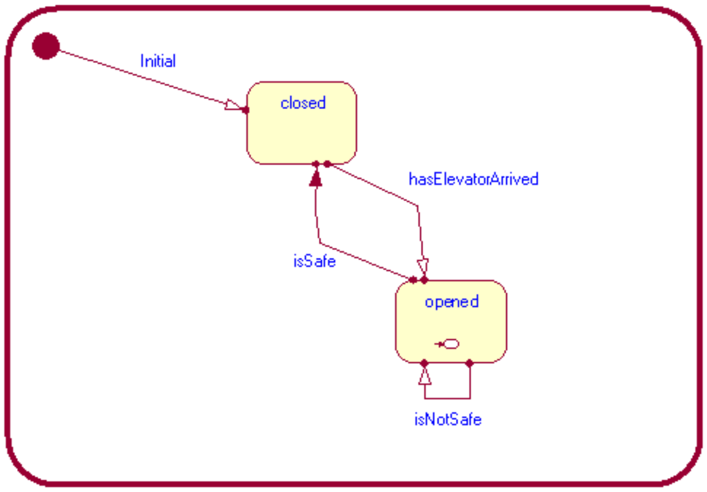

Figure 12: State Diagram for elevatorDoor

The default state in this capsule is called **closed**. When the system receives a trigger via the LocalControl indicating that the elevator has arrived at the specified destination, the **hasElevatorArrived** transition is triggered and it is moved to the **opened** state. There is a self transition in this state called **isNotSafe** which is triggered by a signal from the door sensor indicating that it is not safe to close the door yet. The **isSafe** transition is triggered when a safe signal is received from the door sensor.

3.2.3 Concurrency System

What we have covered so far is how the elevator functions with all its parts, where everything is connected, and how the sequence of events play though as you progress through the system. The only thing we haven't covered yet is how our system handles multiple elevators with multiple request. Earlier in Chapter 3.2.1, we talked about the conditions that the Main Controller checks in order to choose the elevator. Those conditions are: is the elevator moving, which direction the elevator is moving, and is the request on the way. If the elevator is moving, it will then check which direction it is going. If the direction is the same as the direction they wish to travel, then it will check if the elevator has already passed the floor or not. These conditions will cycle through each elevator until one is satisfied with taking the request. How our local controller handles concurrency is due to the ElevatorInfo class, this class is the universal class that is passed around making sure everyone gets the same information. What local Controller does is it has a continue elevator state where, if a new request is sent to that elevator while it is moving. It will not trigger the move function, rather it will just update ElevatorInfo.With this functionality, the Local Controller can still processes the moving while updating where it needs to stop.

3.3 Special Features

3.3.1 Ports and Protocols

Various ports and protocols were used for interactions between the various capsules. The more important protocols are:

elevatorSelect: This was the main protocol for interaction between the main controller and the local controller. It was responsible for relaying important information about the status of the elevators(moving, elevator stopped etc) to the main controller, as well as relaying back the elevator selection to the designated local controller after the selection algorithm is executed inside the main controller.

elevatorMotor: This protocol was used for interaction between the local controller and the elevator capsule. It was mainly responsible for controlling the movement and direction of the elevator itself from the local controller. In return, it also functioned as a means to relay status information about the elevator the the local controller

3.3.2 Passive Classes

A passive class called **elevatorInfo** was used to relay information about the elevator between the different classes. The key variables used in this passive class were **Cfloor** which indicated the current floor, **Dfloor** which indicated the destination floor, **direction** which indicated the direction of movement of the elevator(0 for down, 1 for up), **isMoving** indicates whether or not the elevator is moving(0 for not moving, 1 for moving) and **breaks** indicates whether or not the emergency brake is on. It also consists of a number of getter and setter operations which are used in various capsules.

3.3.3 Dynamic Structures

A passive class called **constants** was created, where the cardinality of the elevators and floors could be set.

3.4 Changes from Milestone 2

We have made many changes since Milestone 2.Figure 13 below shows the capsule diagram from Milestone 2. There were many challenges that the team had to overcome to achieve a functional system.

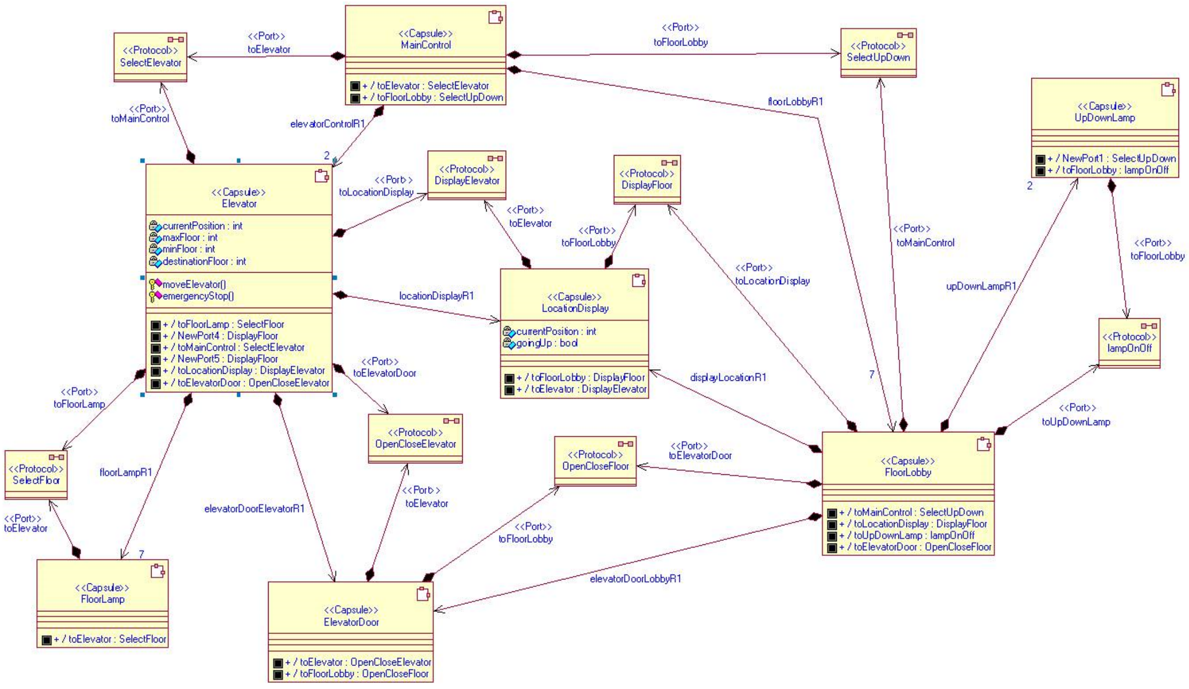

Figure 13: Main Capsule Diagram from Milestone 2

One of the main changes was allowing the local controller to take full control of the doors opening and closing, and adding in emergency breaks. There were also many changes in the state diagrams for the capsules where re-created a more dynamic system rather than a linear system. Figure 14 below shows Milestone 2's dependency diagram.

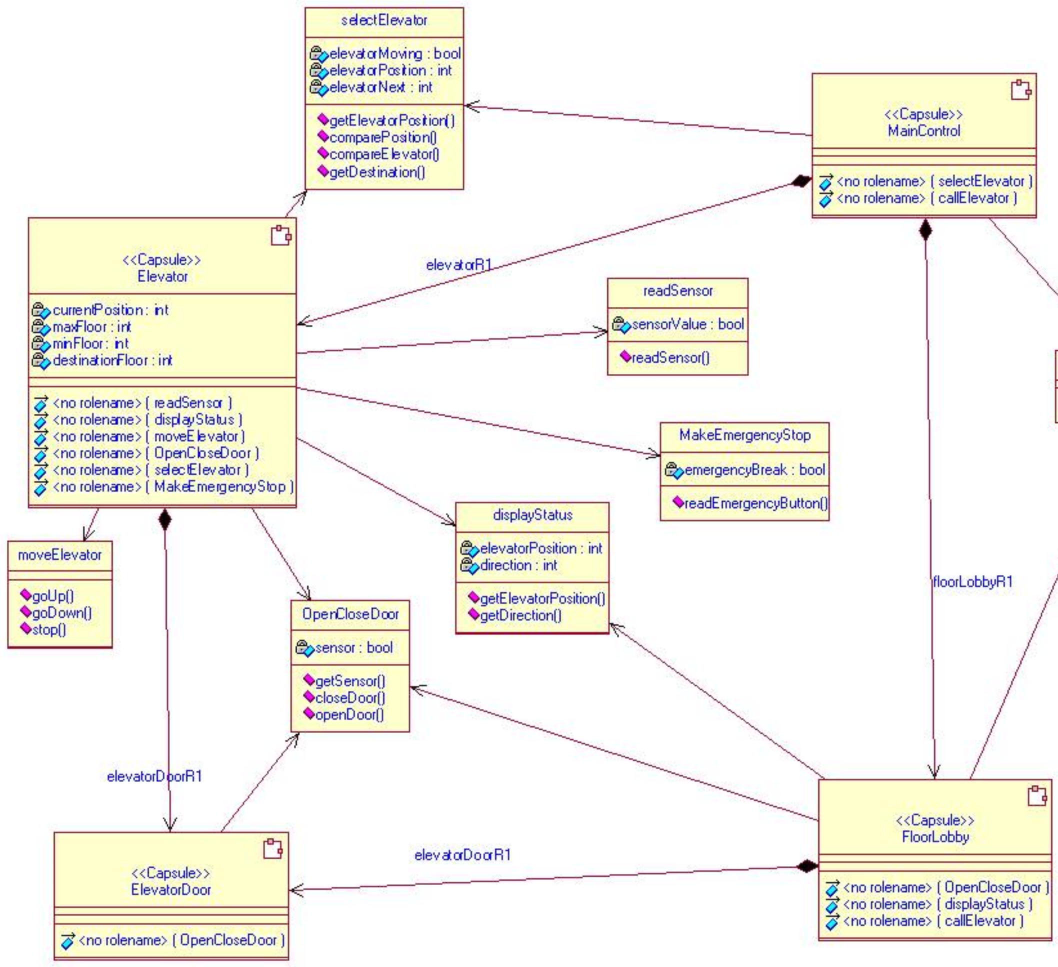

Figure 14: Dependency Diagram from Milestone 2

This diagram was heavily altered going into Milestone 3. The major changes were the passive class that would contain all our constants, and the class ElevatorInfo, which acts as the object that contains all the elevators information. Using the dependency mode, we were able to connect all the major class that use ElevatorInfo in the dependency diagram. This allowed us to handle our concurrecy issues mentioned in Chapter 3.2.3.

Chapter 4: Testing Strategies

4.1 Unit Testing

FloorLobby:

- Signal requestUp for user to request elevator to go to the up direction
- Signal requestDown for user to request elevator to go to the up direction

LocalController:

- Signal moveUp to the elevator motor to move up
- Signal moveDown to the elevator motor to move down
- Signal hasArrived to elevator to inform that elevator has arrived to destination and needs to stop
- Signal NotArrived to elevator to inform the elevator to keep moving to destination

MainController:

- The signal isMoving to select which elevator to go to a specific floor when called by a user depending on different parameters:

DoorSensor:

- Signal SendCurrent to LocalController to update current Location

<u>Other capsules:</u>

DoorSensor:

- Signal safe to close door and it means not obstacle at the door
- Signal is notSafe door remains open because obstacle is present.

EmergencyBrakes:

- Signal releaseBrakes to release brakes and let elevator move
- Signal ApplyBrakes to apply brakes to stop elevator incase of emergency

4.2 Integration Testing

<u>**Testing Scenario:**</u>

1. Probe for destination floor is created from floorLobby capsule for example we could have floor 1 and 6. Since we used cardinality any number of floor could be used.
2. Request up or down signal is sent to mainController from FloorLobby
3. MainController choses elevator depending on different priority (direction where the elevator is moving, location of elevator and if there is a non moving elevator) and sends signal ElevatorSelected to LocalController.
4. LocalController sends signal MoveUp or Down to the selected Elevator to move elevatorMotor and elevator moves to desired floor depending on the floor being chosen.
5. Elevator sends signal SendCurrent to local controller to inform about the current location
6. Function CheckFloor check if elevator has arrived to destination and sends two signals to elevator

DoorSensor:

a. Arrived: signal ElevatorDoor to open door

    i. A probe is created to send signal Safe to doors to close

ii. If not safe, NotSafe signals is sent and doors stay open

iii. Elevator could stop at multiple floors depending how many users are in the elvator.

b. NotArrived: signal LocalController MoveUp/Down depending on destination.

7. Door signal ElevatorStopped to Localcontroller to set the status of elevator to ready
8. If there is more than one destination, ElevatorCheck signal makes sure the elevator gets to all desired destination.

EmergencyBrakes:

9. A probe is created to signal EmergencyBrakes while the elevator is moving to stop elevator and the signal is called applyBrakes which is sent to localController
10. Another probe is created to signal EmergencyBrakes while the elevator is stopped to move elevator and the signal is called releaseBrakes which is sent to localController.

4.3 System behavior coverage:

Happy Path(s)

Case 1

1. User presses button to go up/down outside elevator.
2. User enters the elevator selects floor
3. User waits for 10 seconds for door to close
4. User arrives at required floor and waits for doors to open

5. User exits

Case 2

1. User presses button to go up/down outside elevator.
2. User enters the elevator and selects floor
3. User waits for 10 seconds for door to close
4. Obstruction present at the door
5. Sensor detects obstruction at door and keeps door open for 10 more seconds
6. Obstruction is cleared from the door
7. Sensor detects door is clear and closes door
8. User arrives at required floor and waits for doors to open
9. User exits

Case 4

1. User presses button to go up/down outside the elevator
2. User enters the elevator and selects floor.
3. User waits for 10 seconds for door to close
4. User presses on a button.
5. Button does not light up or get selected.
6. User presses on Emergency button
7. Elevator stops working and doors stay open allowing users to leave.

Case 5

1. User presses button to go up/down outside elevator.
2. User enters the elevator selects floor

3. User waits for 10 seconds for door to close
4. Elevator does not stop on the desired floor
5. User presses emergency stop button
6. Emergency brakes applied
7. Elevator stops at the closest floor.

Unhappy Path(s)

Case 1

1. User presses button to go up/down.
2. User enters the elevator selects floor
3. User waits for 10 seconds for door to close
4. Door does not close
5. User presses on emergency button
6. Emergency stop does not function.
7. The ElevatorController stops the elevator and keeps the doors open

Case 2

1. User presses button to go up/down outside elevator.
2. User enters the elevator selects floor
3. User waits for 10 seconds for door to close
4. Elevator does not stop on the desired floor
5. User presses emergency stop button

6. Emergency stop does not function
7. The ElevatorController stops the elevator and keeps the doors open

Case 3

1. User presses button to go up/down outside elevator.
2. User enters the elevator and selects floor
3. User waits for 10 seconds for door to close
4. Obstruction present at the door
5. Sensor does not detect obstructions
6. Doors keep closing.
7. The ElevatorController stops the elevator and keeps the doors open

4.4 Checklist of General Tests

Outside Elevator

1) You can call elevator from every floor

2) When elevator arrives the light outside shows which direction the elevator is going too.

3) The direction indicators disappears when its last occupants are delivered to a floor.

4) The elevator doors open within a reasonable amount of time after arrival

5) The elevator doors remain open within a reasonable amount of time

6) Elevator doors reopen when blocked during closure

7) Elevator doors close in a reasonable when it is safe and when user goes in or out.

Inside Elevator

1) pressing all buttons from bottom floor causes elevator to stop at each ascending floor in succession.

2) Pressing all buttons from top floor causes elevator to stop at each descending floor in succession.

3) Pressing only top floor button from bottom floor causes elevator to deliver you to the top floor without stopping if no one is going up in the middle floors.

4) Pressing only bottom floor button from top floor causes elevator to deliver you to the bottom floor without stopping if no one is going down in the middle floors.

5) Elevator travels in the same direction until all illuminated buttons have extinguished in that direction.

6) Elevator changes direction when there are no more illuminated buttons in that direction.

7) Elevator shows which floor you are in on the display

8) Direction in which the elevator is going is showed.

9) Floor numbers are accurate and clearly visible upon reaching their corresponding floors.

PERFORMANCE TESTS

1) The amount of time (speed) elevator travels between floors meets specification.

2) The amount of time elevator waits between opening and closing doors meets specification.

3) the amount of time the elevator takes to get to the desired floor meets the specifications

4) the elevator closest to the floor and going to the same direction as the desired floor stops there to save time.

4.5 Traces

4.5.1 Trace of one elevator moving up and doors opening/closing

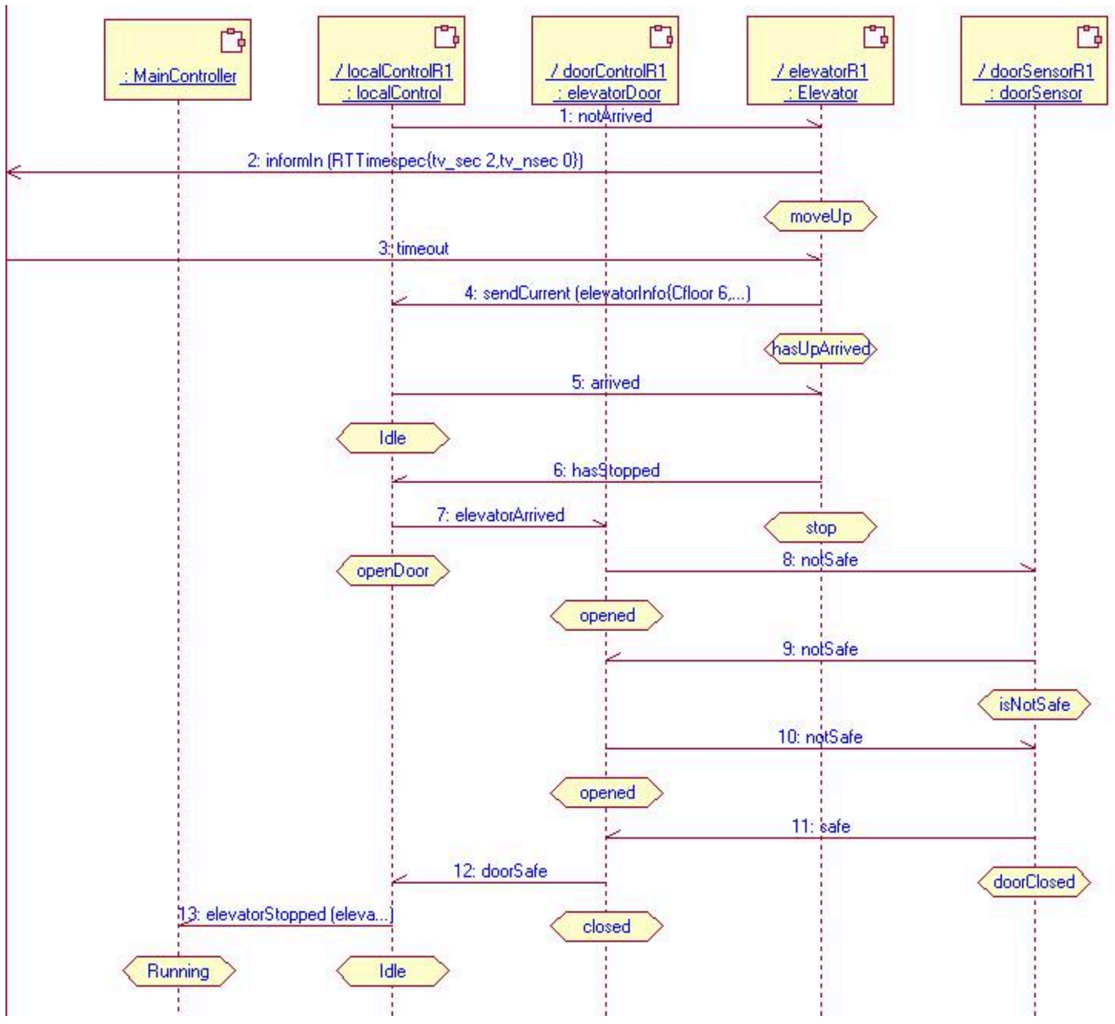

Figure 15: elevator moving up and doors functioning

4.5.2 Trace of one elevator moving Down and the other elevator moving up

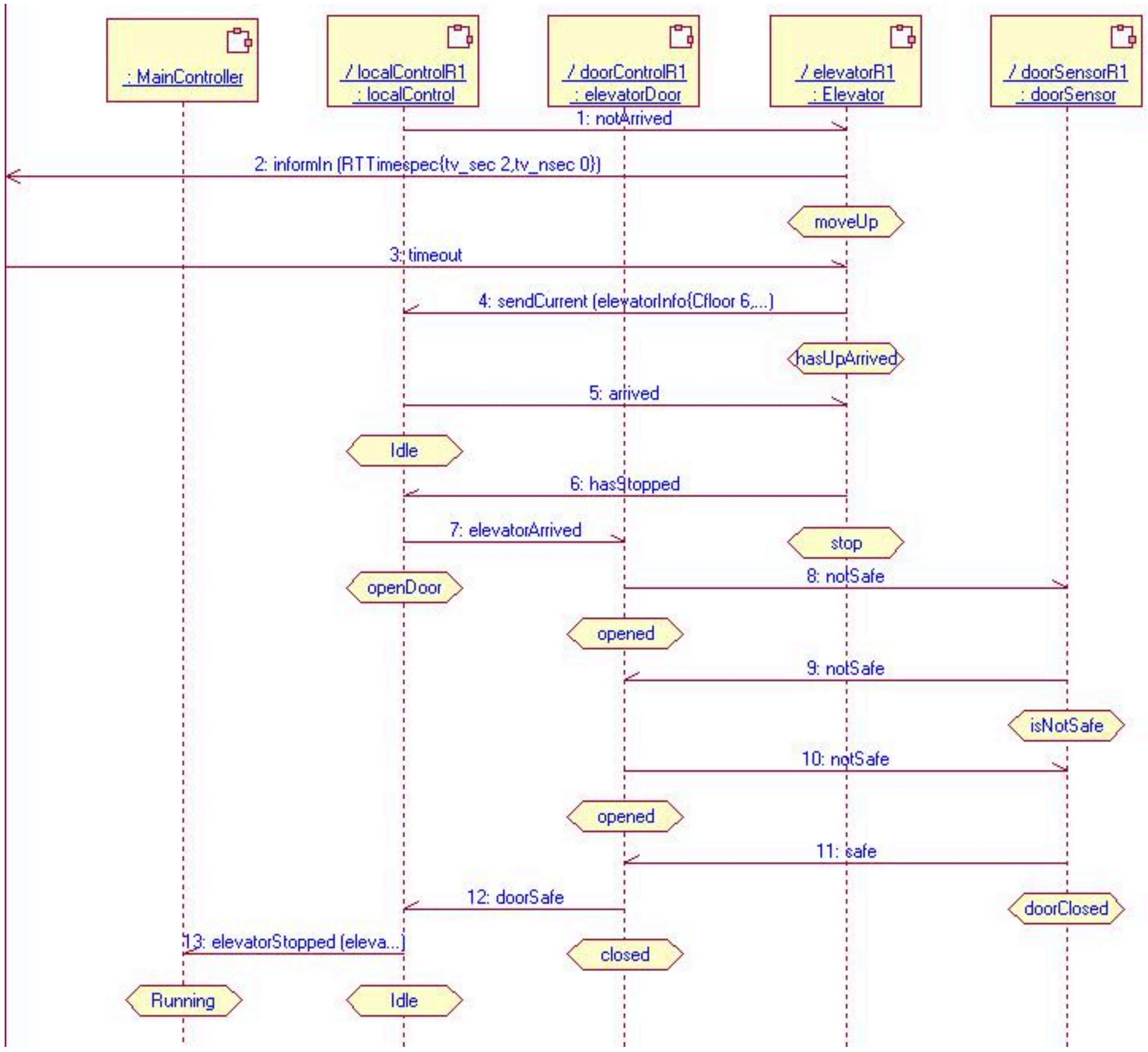

Figure 16: two elevators moving up and down at the same time

4.5.3 Trace of emergency brakes being applied

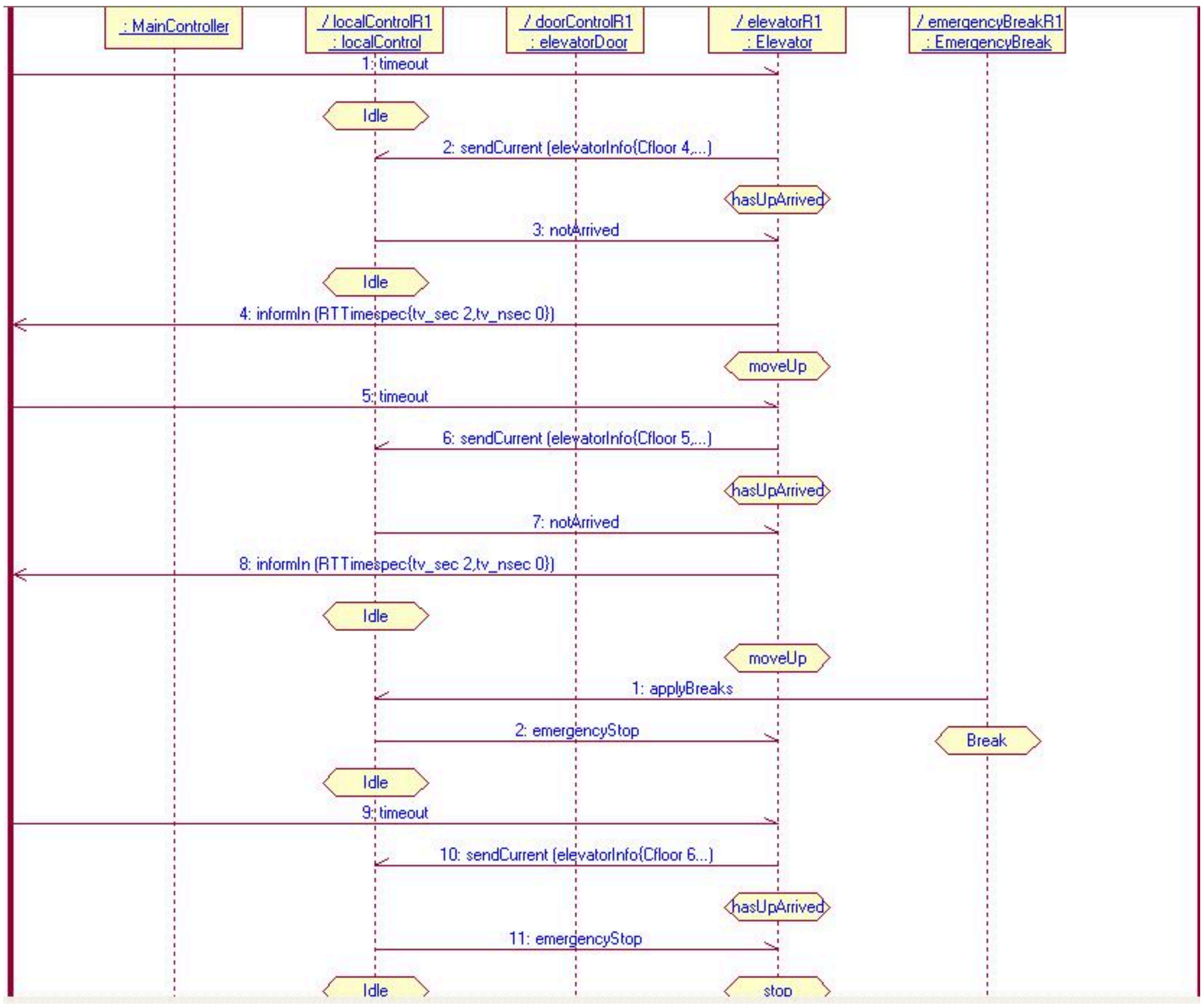

Figure 17: Emergency brakes being applied

Chapter 5: Conclusion

UML-RT is a program for developing and simulating real world applications. These types of simulations help the user and the designer to have a better understanding on how the system will function in a real life situation. The reasons to use simulations may be many, like to estimate cost, time, or for safety reasons. Moreover, this helps buyers to have better knowledge of the product before purchasing it and allows them to see if it caters to their needs. This is where designers settle on an Object-Oriented solution to simulate such projects. In case of this project, it was to develop, simulate and test the selection and working of an elevator system. The efficient simulation of this system may allow a suitable company to implement design ideas which may end up providing economic gain and profits.