CS846: Requirements Engineering

Software Requirements Specification (SRS):

Elevator Controller for low-rise building elevator system

Specification Version: *Completed SRS 1.1*

Prepared for Daniel M. Berry

University of Waterloo

CS846 (Fall 2005)

Prepared by: **Joel So** *(j2so@cs.uwaterloo.ca)*

Last Modified: **November 24, 2005**

Table of Contents

List of Figures

List of Tables

1 Introduction

This document is the complete product requirement specification for an elevator controller software system for low-rise building elevator systems. Henceforth, this is the only document that contains all information regarding the requirements placed on this elevator controller by the stakeholders, catalogued in an unambiguous fashion. Unless otherwise stated, this document, and any future revisions of this document, supersedes all other requirements documents that exist for the said elevator controller system.

1.1 Purpose

This is a Software Requirements Specification (SRS) document. The purpose of this document is two-fold:

1. To capture and organize functional and non-functional requirements for an elevator controller software system (for low-rise building elevator systems).
2. To formally specify the intended behaviour and functionality of the elevator controller software system so that it addresses the functional and non-functional requirements.

The intended audience for this SRS is software engineers implementing the specified elevator controller and stakeholders of either the elevator controller itself or the elevator system/building where the controller will be deployed. The reader is assumed to have general knowledge on the basic theory and operation of a regular elevator system (in a low-rise building), and have some experience with the terminology used in the document. Also, experience in reading UML 2.0 diagrams is required.

Once finalized and signed-off by all stakeholders, this document serves as a contract between stakeholders as to what is expected of the elevator controller, and how the components of the elevator controller are to work with each other and with external systems (the elevator system at large).

1.2 Scope

This SRS specifies an elevator controller software system for low-rise building elevator systems. An elevator controller controls and choreographs elevator hardware components (cars, sheave motors, doors, etc.) so that elevator passengers can be transported between floors of a building. An elevator controller must interface with various input/output components that interact with elevator users (passengers and operators).

The elevator controller specified here is intended specifically for low-rise buildings. The specificity to low-rise buildings results in various (simplifying) assumptions about the requirements and behaviour of the elevator controller. These are listed in Section 2.5.

This document covers the details of the elevator controller system (the system), including the physical and related components of the system, and the behavioral, functional, and non-functional requirements. This document describes only the external systems and/or environments in which the elevator controller system shall work, with enough detail to complete an implementation of the same.

1.3 Acronyms, Abbreviations, Definitions, Notation

Deluminate

The opposite of *illuminate*. To turn off the light of something that is lighted.

EC

Elevator Controller. A software system used to control and choreograph elevator hardware components (cars, sheave motors, doors, etc.) so that elevator passengers can be transported between floors of a building.

Sheave

A pulley with grooves around the circumference. In a cable-driven elevator system, elevator cables attached to the elevator car are wrapped around sheaves at the top of the elevator shaft. Motors rotate the sheaves thus pulling/releasing the elevator cables (via static friction between the grooved pulley and cable) and raising/lowering the elevator car.

Sheave Motor

An electric motor which rotates a sheave (either directly, or by way of a gearing system).

1.4 References

Document/Resource Name	Location (URL/Contact)
[1] HowStuffWorks.com: How Elevators Work	http://science.howstuffworks.com/elevator.htm
[2] Project Outline	http://se.uwaterloo.ca/~dberry/ATRE/project.html
[3] Stakeholder Contact: Davor Svetinovic	dsvetinovic@swag.uwaterloo.ca, davorss@gmail.com
[4] Observational Analysis & Assumptions of Real-World Elevator Behaviour	j2so@cs.uwaterloo.ca
[5] Example SRS (Turnstile Controller)	http://se.uwaterloo.ca/~dberry/ATRE/uberTurnstile_srs.pdf

Table 1: Table of references.

1.5 Overview

In the rest of this document, Section 2 gives a general description of the system and identifies all assumptions and constraints placed upon the system. Section 3 exposes the specific requirements of the system demanded by stakeholders and their relationships via various UML 2.0 diagrams. Finally, Section 4 provides reference information including use cases, functional and non-functional requirements tables, and traceability information.

2 General Description

Low-rise buildings typically have three or more floors. To assist building residents and visitors in traversing these floors, simple two-car, cable-driven elevator systems are often installed. As with all modern elevators, an **elevator controller** (embedded software/firmware system) is required to control the movement and operation of the two elevators. This SRS document specifies an elevator controller *specifically for two-car low-rise building elevator systems.*

2.1 Product Perspective

Typical elevator hardware (shafts, cars, cables, floor doors, etc.) and hardware-building configuration is assumed. The following figures highlight (example) components visible to typical elevator *end-users* (passengers and operators). Some of these components will interface with the elevator controller. A six-floor low-rise building is used for example.

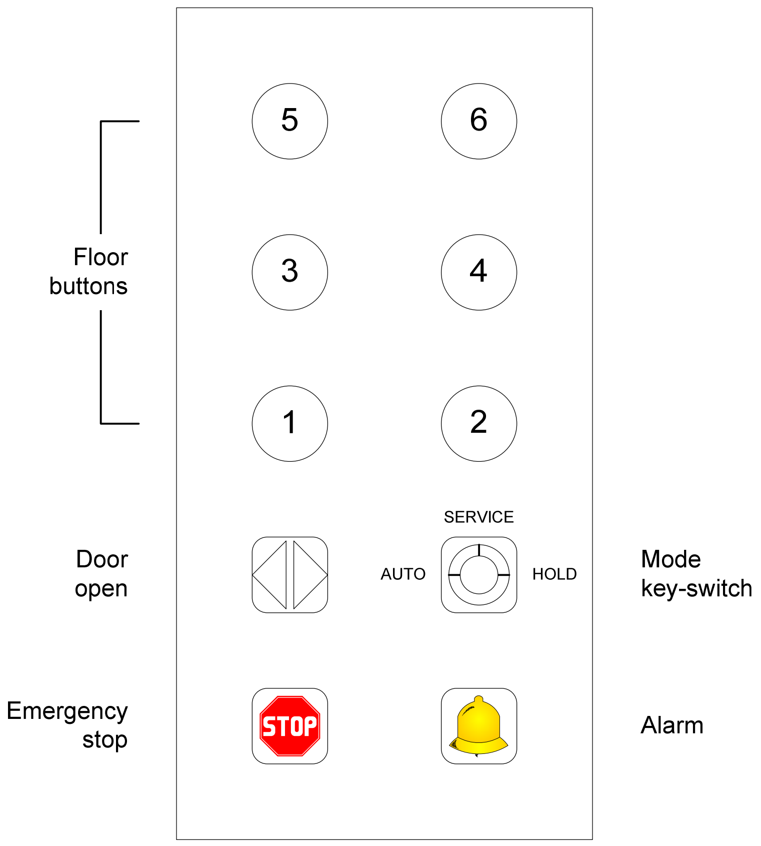

Figure 1: Example in-car button panel.

Figure 1 shows a typical button panel inside each elevator car. The button panel has the following buttons/switches: floor buttons 1 to 6, door open button, emergency stop button, alarm button, and elevator mode key-switch (AUTO/SERVICE/HOLD). Elevator modes, emergency stop behaviour, and alarm behaviour are detailed in subsequent sections.

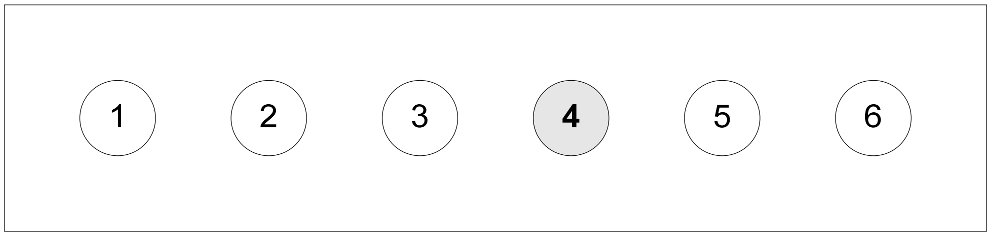

Figure 2: Example floor indicator.

Figure 2 shows a typical floor indicator panel installed above each elevator door on each floor of the building, and above the door inside each elevator car. The display panel illuminates the number corresponding to the current position (floor) of the elevator car that it represents.

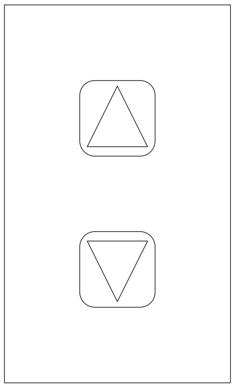

Figure 3: Up/down button panel.

Figure 3 shows a typical up/down button panel installed at each elevator entrance located on each floor of the building. The up/down button panel is used by passengers awaiting pick-up to indicate their desired direction of travel. Each elevator entrance has two elevator doors, one for each elevator car, on either side of the up/down button panel. There is only one up/down button panel per elevator entrance.

For all intents and purposes, it is assumed that the elevator entrances on all floors are identical to one another. Likewise, the two elevator cars are also assumed to be identical to one another.

Note: Refer to Section 3.2.2 for complete specification of signals transmitted and commands received by elevator components (i.e. the hardware application programming interface).

The purpose of the elevator controller is to control elevator hardware components (cars, sheave motors, doors, indicators, etc.). The elevator controller (**EC**) is a software system that is (presumably) flash-loaded as firmware on a hardware controller device located in the building's elevator control room. Once running, the EC accepts input signals sent to the controller device from other elevator components, and outputs signals to elevator components to control their behaviour. The EC is a sub-system of the overall elevator system (the high-level construct with which human users, passengers and operators, interface). As a lower-level sub-system, the EC has a different context and set of interfaces than that of the overall elevator system. Thus, the EC context is as follows:

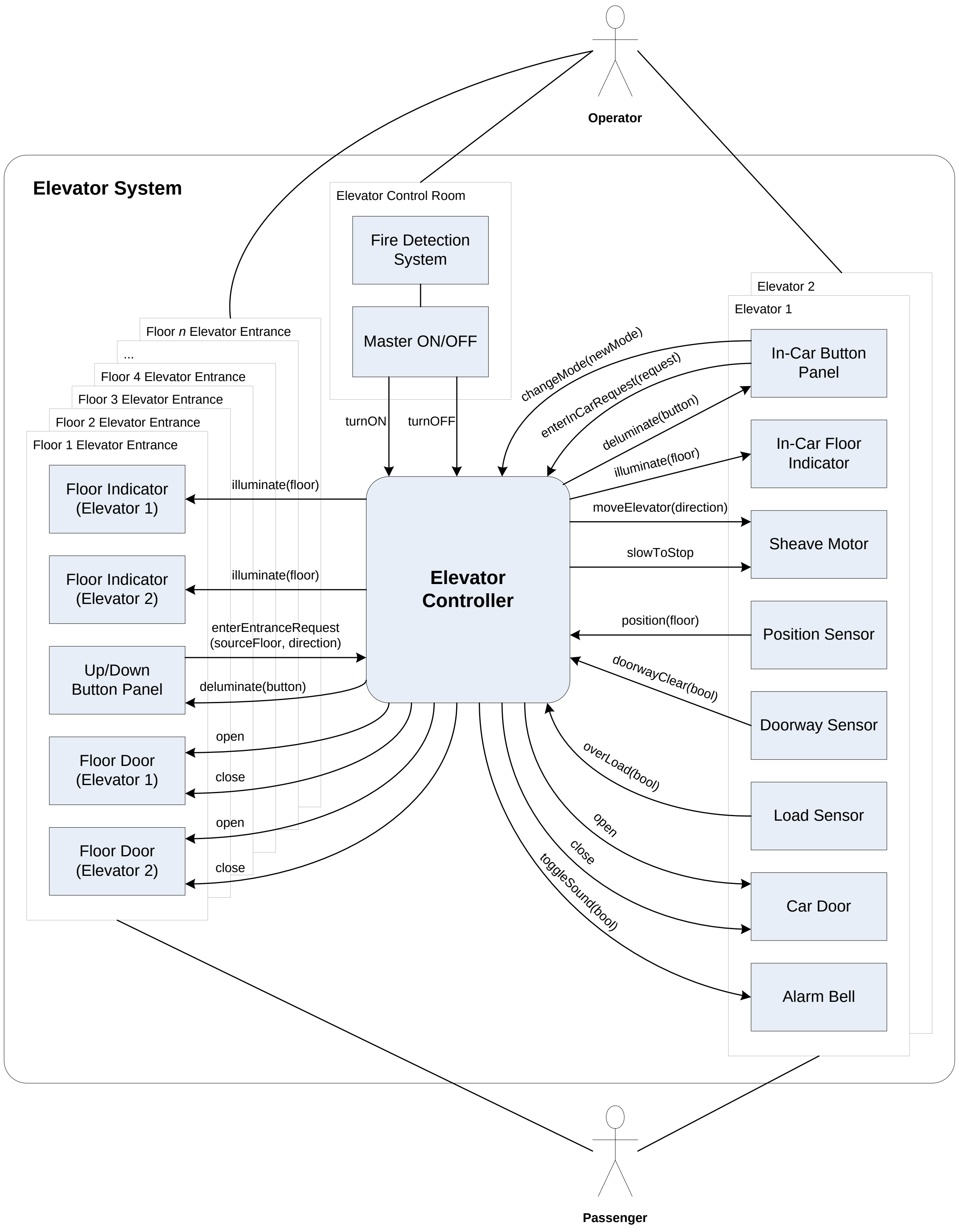

Figure 4: Elevator Controller Context Diagram.

2.2 Product Functions

The primary function of the EC is to control the up and down movement of the elevator cars (by way of controlling the elevator sheave motors). It does so to position the elevator cars at floors where passengers have selected (either for pick-up or drop-off). It also controls the opening and closing of elevator car doors and floor entrance doors to allow the *safe* entry and exit of passengers into and out of the elevator cars.

In addition, the EC controls illumination and delumination of floor indicators and buttons.

The EC supports **two operational modes** and **one recall mode**: AUTO, HOLD, and SERVICE. The specific mode in which the EC is operating determines the way in which it controls the up-down movement of the cars and open-close actions of the doors. The mode is selected per elevator car using the mode key-switch on in-car button panels. Behaviour of each mode is as follows:

- In **AUTO** mode (operational), the elevator behaves as a typical elevator would. Elevator cars are sent to floors where pick-up or drop-off requests have been made (via up/down button panels at each elevator entrance or in-car button panels, respecticely)
- In **HOLD** mode (operational), the elevator behaves as a service elevator, ignoring any passenger pick-up requests, but instead changing floors only when floor selection is made by a passenger *inside* the elevator car (using the in-car button panel). While in HOLD mode, the elevator doors stay open indefinitely at each destination floor (doors are closed while the elevator is in motion, of course). Only one destination floor can be selected at a time while in this mode. HOLD mode is generally used to facilitate move-ins (i.e. someone moving into a building) or planned transportation of large items.
- In **SERVICE** mode (recall), the elevator is returned to a (pre-configured) default/recall floor and remains on that floor with the doors open. Reanimation of the stopped elevator car then requires operator action (operator must use the in-car mode key switch to change to one of the two operational modes).

2.3 User Characteristics

There are two types of eventual end-users of the *overall* elevator system: **passengers** and **operators**.

A **passenger** is anyone who wishes to enter an elevator car on floor ***i***, select a destination floor ***j*** (1 <= ***i, j*** <= # of floors, ***i*** != ***j***), “ride” the elevator car until it arrives at floor ***j***, and exit the elevator car. A passenger is expected to be able use up/down buttons to request elevator pick-up, to enter and exit an elevator car through elevator doors, and to select desired destination floor(s) by pressing buttons on in-car button panels.

An **operator** is someone who performs any of the following:

- Changes the elevator mode (AUTO/ HOLD/SERVICE) of an elevator car using the key-switch on in-car button panels.
- Turns the elevator system ON and OFF using a key-switch in the elevator control room.
- Flash-loads new EC firmware onto the controller device (**Note:** let us consider this action as *out-of-scope* for this specification – it should be in-scope for a specification of the controller device hardware).

An operator is expected to know how to use a key-switch (given the correct key) and understand the consequences of her configuration choices (made via key-switch). **An operator is also a passenger.**

2.4 General Constraints

1. The elevator hardware (including human-interface components) is already in place and is fixed. Therefore, the EC must not imply/require any change in elevator hardware.
2. General functionality of overall elevator system (not to be confused with EC sub-system) is already established and is fixed (e.g. the two operational modes, the recall mode, the *expected* sequence of illumination/delumination of buttons or indicators, etc.). Therefore, the EC must not redefine such generally understood/expected features or behaviours.

2.5 Assumptions & Dependencies

2.5.1 General Assumptions

1. The EC specified in this SRS is done so *specifically* for low-rise building elevator systems, and all existing hardware, number of floors, cars, shafts, etc. This assumption also has the following implications:
    a. Extensibility of the EC (e.g. to support high-rise elevator systems and/or more than two cars, etc.) is *not* a requirement (or is to be considered as part of the software design, not the requirements specification)
    b. In a low-rise setting, the EC need not implement any complex optimization techniques for scheduling car movement. Car selection for passenger pick-up will simply be determined by which car is closest for passenger pick-up (and if already traveling, is going in the logically correct direction), and cars will simply wait at the last-stopped floor for more passenger requests (as opposed to, say, going to a floor that is statistically most likely to have the next passenger request). There will not be any logging of usage statistics.
2. The following assumptions are made regarding the *desired/standard* behaviour of low-rise building elevator systems:
    a. Each elevator car is equipped with an **alarm bell**. When the alarm button of an in-car button panel is *pressed-and-held*, the alarm bell for that car is set to sound and continues to sound until the button is released. While the alarm bell is sounding, the elevator will continue operating as usual. Therefore, the purpose of the alarm is merely to notify elevator operators that attention is required (for whatever reason) – it does *not* imply that the elevator must be halted.
    b. The **emergency stop button** of an in-car button panel is intended to give passengers the option of returning the elevator car to a default recall floor immediately in case of emergency. When this button is pressed, it is assumed that the elevator car is to go to a (pre-configured) default/recall floor and open the elevator doors. Reanimation of the stopped elevator car then requires operator action (operator must use the in-car mode key switch to reset the stopped car or restart the entire elevator system). Notice that

pressing the emergency stop button is functionally equivalent to changing the elevator to the recall SERVICE mode (using the mode key-switch).

c. It is assumed that a building's **Fire Detection System** does have some control over the elevator system in the event that a fire is detected. For simplicity, it is assumed that the Fire Detection System simply triggers an elevator system shut-down in the event of a fire.

d. It is assumed that the elevators are to be returned to a (pre-configured) default/recall floor upon system shut-down, and left on that floor with all elevator doors open. For simplicity, it is also assumed that elevators are to start on the same default/recall floor upon system start-up (elevator cars will be returned to the default/recall floor as part of the start-up sequence in case they were somehow manually relocated while in the OFF state).

3. A single EC instance controls both elevators cars and related components.

2.5.2 Assumptions on Product Scope

1. Deployment of EC firmware (i.e. flashing a controller device) is out-of-scope (this should be considered in a requirements specification for the controller device hardware).
2. The emergency phone system (to which the phone device in each elevator car is connected) is unrelated to and not controlled by the EC.
3. Any emergency braking system that the elevator cars may have is entirely mechanical (including mechanical triggering/engagement) and is not controlled by the EC.
4. The elevator car ventilation and lighting systems (e.g. controlled by keyed-switches in each elevator car) are independent of and not controlled by the EC (presumably they are both implemented by way of simple hard-wired circuit switches).

2.5.3 Assumptions on Existing Elevator Hardware & Interfaces

1. Each hardware component that interfaces/interacts with the EC does so by sending and receiving low-level electrical signals to and from the controller device hardware. It is assumed that the EC software can treat these signals as processed/interpreted signals (perhaps through some hardware abstraction layer implemented in the controller device). As such, logical labels/names and parameters can be used to describe such signals. For example, the EC can send a logical "open" or "close" signal to a door component. Or, a button panel can send a "enterRequest(*request*)" signal to the EC indicating that the specific button *request* was pushed. A complete list of assumed signal types and parameters are provided in Section 3.2.2. The general assumption here is that such a logical abstraction of signals/parameters exists and that, for all intents and purposes of this SRS, the logical labels or descriptors can be used to describe the signaling between the EC and other elevator components.
2. All modern cable-driven elevator systems make use of electro-magnetic brakes to hold the car in a position once the sheave motor has stopped turning. It is assumed that the actions of these electro-magnetic brakes are implicit and in unison with the actions of the sheave motor. That is, when the sheave motor is turning, the brakes are implicitly disengaged, and once the sheave motor has come to a stop, the brakes are implicitly engaged. This is a reasonable assumption as electro-magnetic brakes actually engage with zero voltage passing through, and disengage with positive

voltage passing through (say, the same voltage passing through the sheave motor to make it turn). The implication here is that the EC need not control the electro-magnetic brakes explicitly – controlling the sheave motor implicitly controls the brakes.

3. Gradual acceleration and deceleration of the sheave motor (and in turn, the elevator car) is done implicitly. That is, the EC can simply signal the sheave motor to start, and it will do so gradually (so as not to scare or rattle passengers in the elevator car). Similarly, when the sheave motor is signaled to stop, it does so gradually, slowing to a stop. It is also assumed that the delay in reaching full stop from the time a stop signal is issued (due to the gradual deceleration) is accounted for by skew in position sensor signals. That is, if a position sensor indicates to the EC that the elevator car is positioned at floor ***i***, it really means that if the EC signaled the sheave motor to stop at that instance, that the elevator car would be accurately positioned at floor ***i*** after traveling the additional distance required to come to a full stop.
4. Whenever a floor button on the in-car button panel, or any button on the up/down button panels, is pushed, it automatically illuminates instantly and stays illuminated until signaled to deluminate by the EC. The EC does not control the illumination of these buttons, only the delumination.
5. When the EC signals a door to close, it is guaranteed to actually close within some fixed period of time (say, 3 seconds) unless a doorway block is detected. If no doorway blockages are detected within the fixed period of time after a door has been signaled to close, the door is assumed to be actually closed. Doors need not signal the EC to confirm closed status.
6. Elevator hardware is assumed to never fail, or rather, that failure detection is not the job of the EC. That is, it is assumed that, from the perspective of the EC, all components that it interfaces with are working. If any are not working correctly, it will not change the behaviour of the EC (though it may change the behaviour of the overall elevator system). The EC continues under the assumption that components are receiving and sending signals correctly, and acting accordingly. If hardware failures occur, it is up to an operator to detect it (or be notified of it) and shut down the EC and elevator system manually. The EC does not implement any diagnostics routines or features.

3 Specific Requirements

The following sections formally specify EC functionality (using UML 2.0 where appropriate) and external interface requirements. Functional specification is derived from the use cases and functional requirements listed in Sections 4.1 and 4.3. The hardware interface specification extends the first assumption listed in Section 2.5.3 to specify a complete list of assumed signals and parameters.

3.1 Functional Requirements

3.1.1 Overall System

The following packaged use case diagram groups use cases into related concerns.

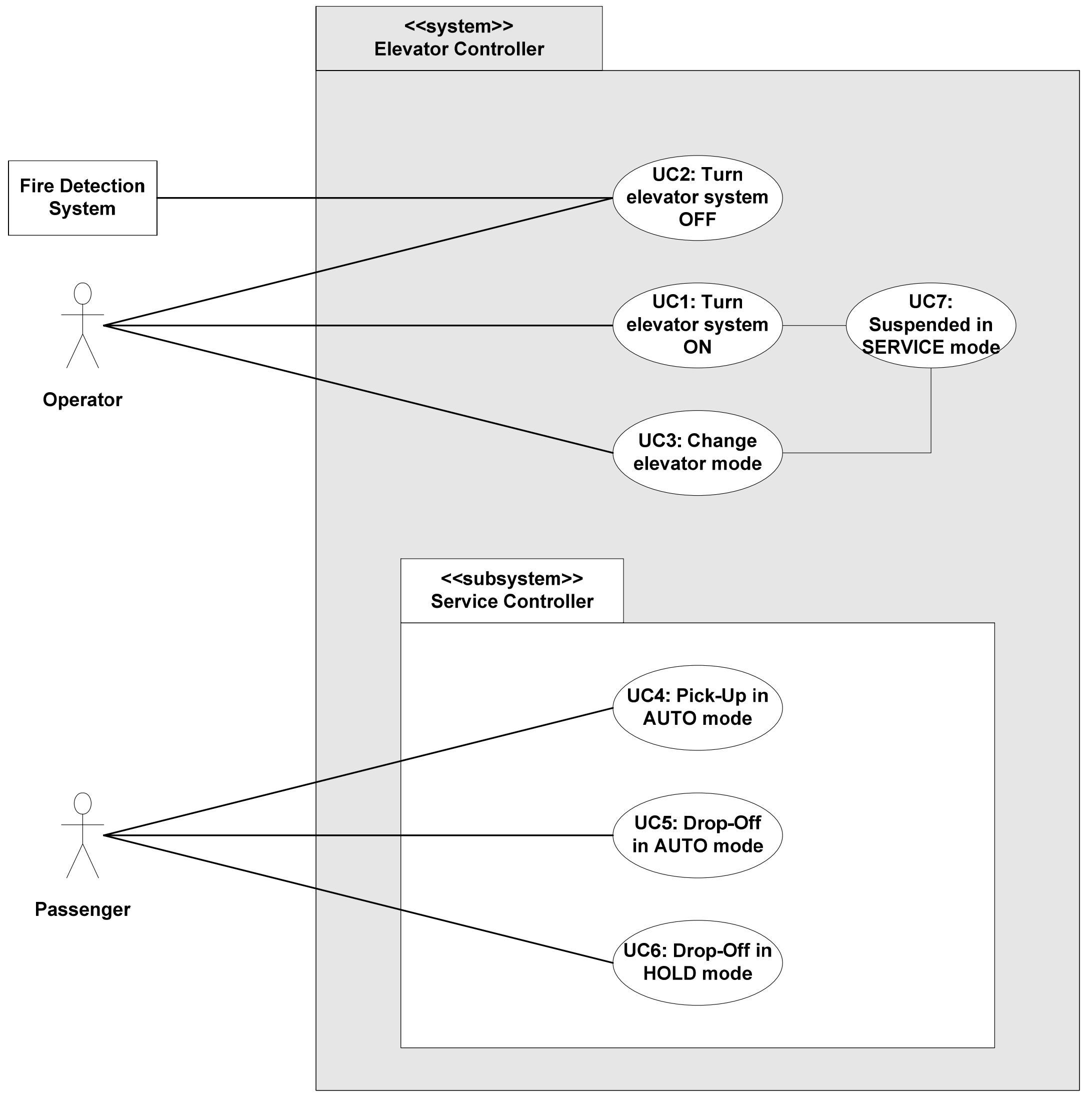
<<system>>
Elevator Controller
Fire Detection System
Operator
UC2: Turn elevator system OFF
UC1: Turn elevator system ON
UC7: Suspended in SERVICE mode
UC3: Change elevator mode
<<subsystem>>
Service Controller
UC4: Pick-Up in AUTO mode
UC5: Drop-Off in AUTO mode
UC6: Drop-Off in HOLD mode
Passenger

Figure 5: Functional subsystem grouping of use cases.

3.1.1.1 System Sequence Diagrams

The following sequence diagrams illustrate actor/user interaction with the EC system and subsystems based on the use cases listed in Section 4.3.

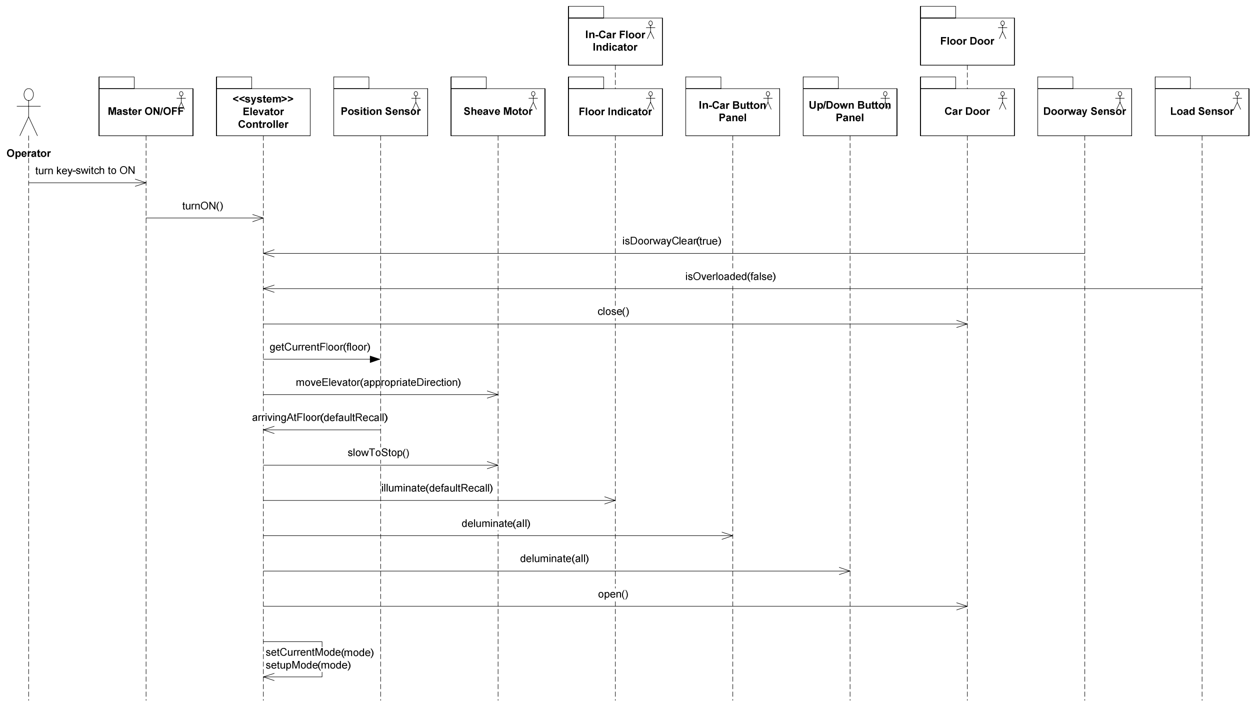

Figure 6: UC1: Turn elevator system ON

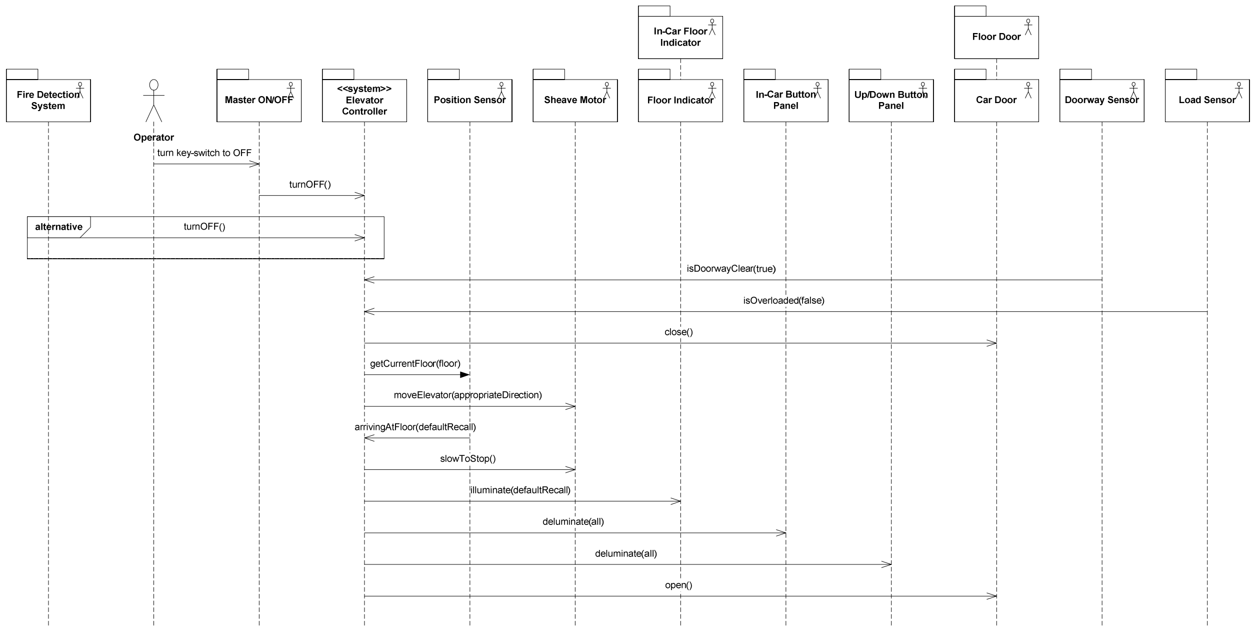

Figure 7: UC2: Turn elevator system OFF

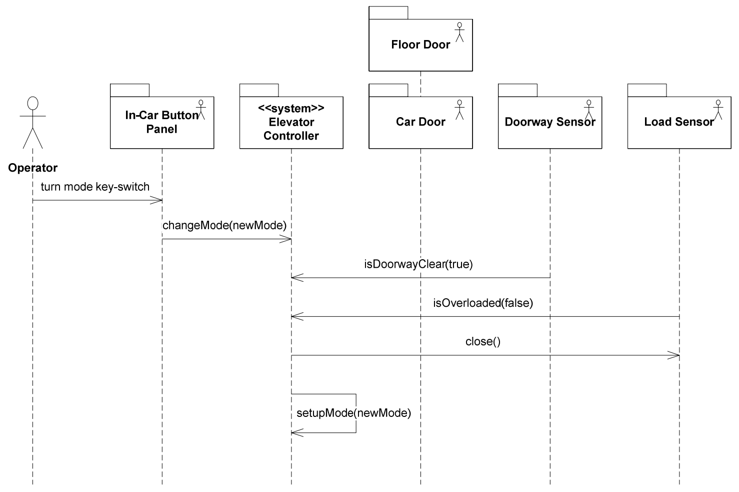

Figure 8: UC3: Change elevator mode

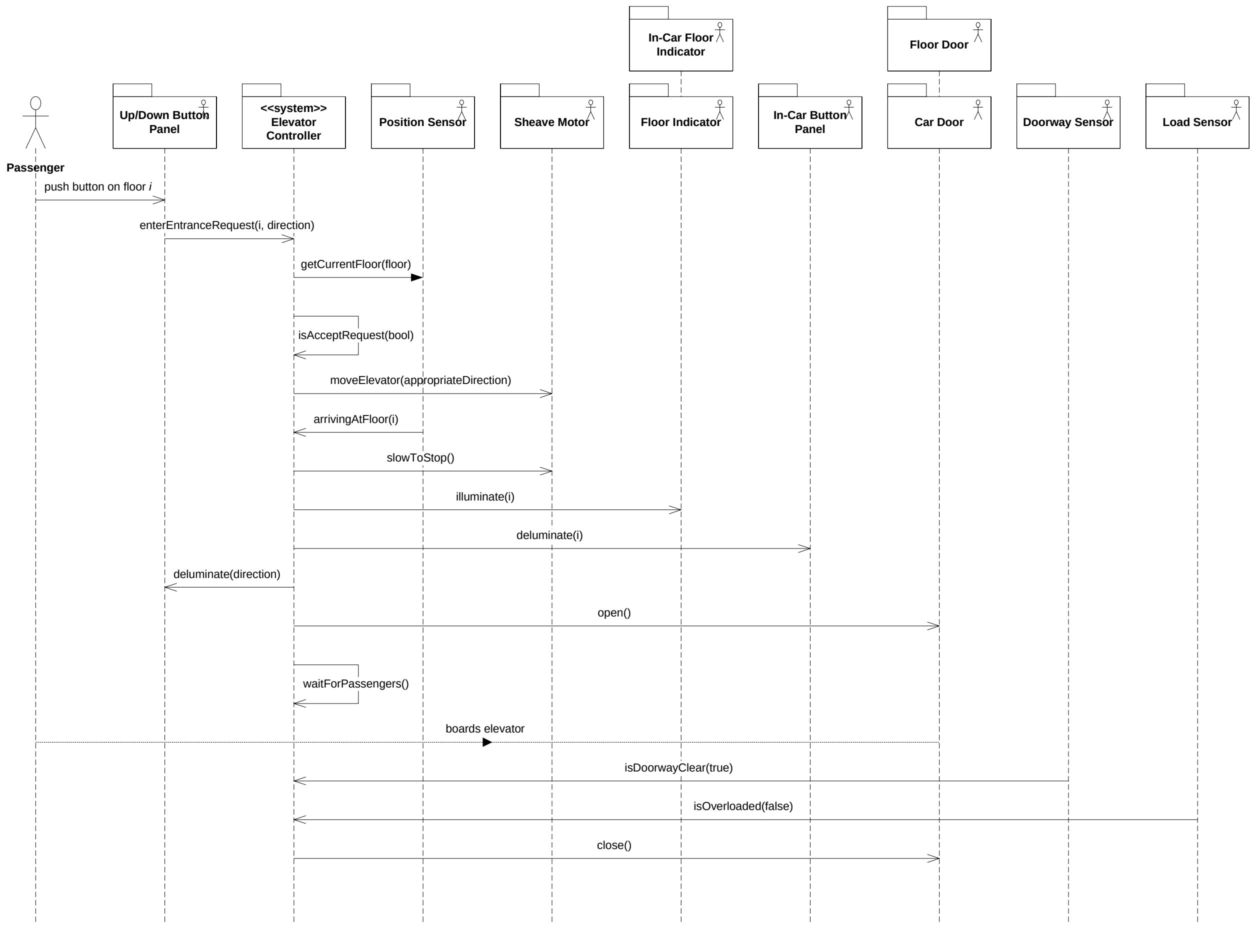

Figure 9: UC4: Pick-Up in AUTO mode

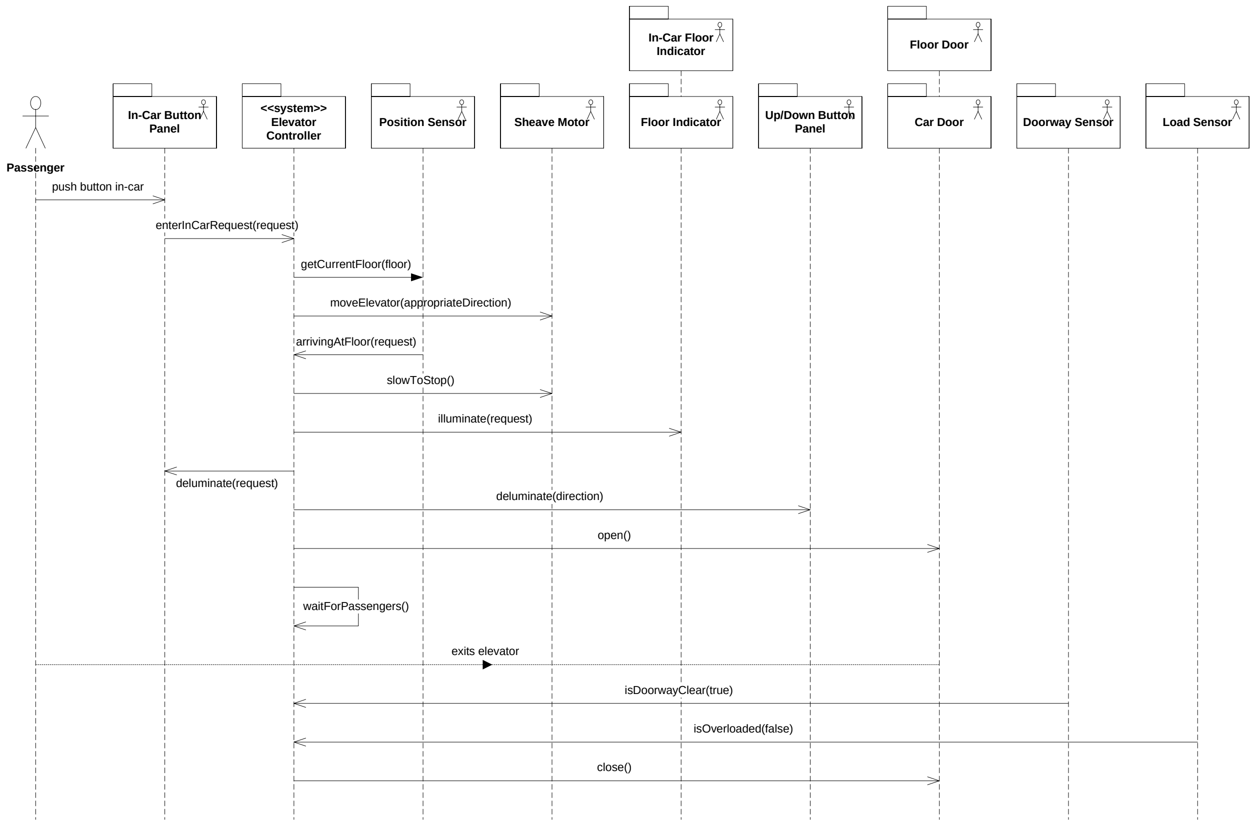

Figure 10: UC5: Drop-Off in AUTO mode

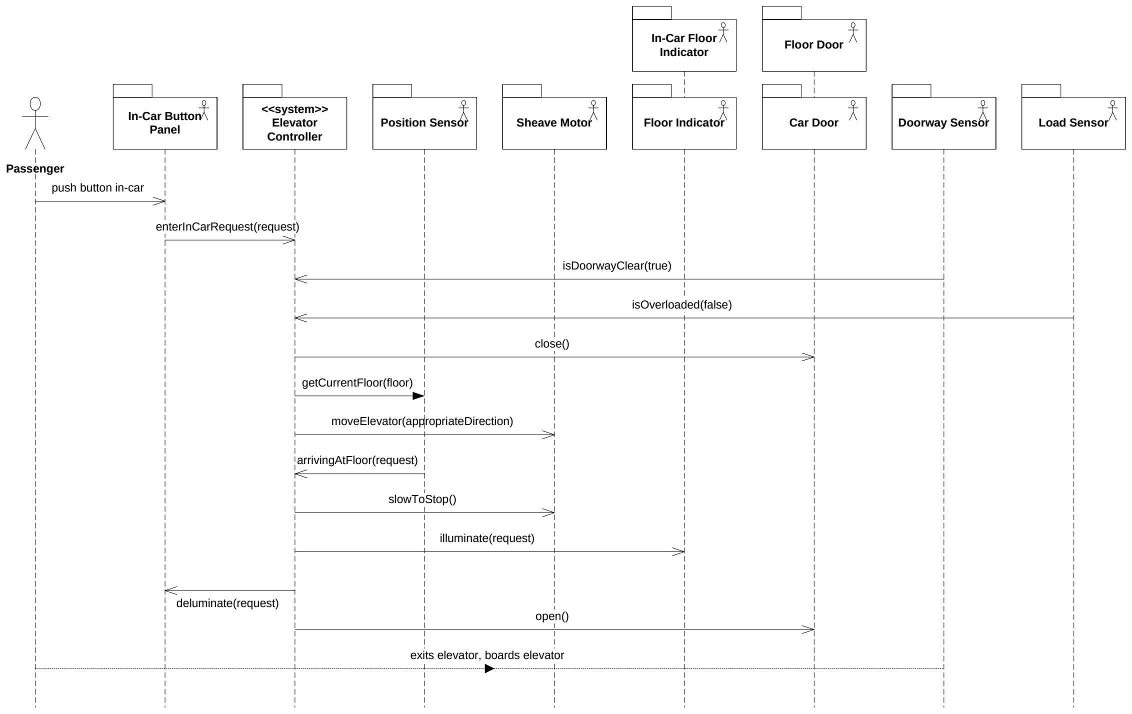

Figure 11: UC6: Drop-Off in HOLD mode

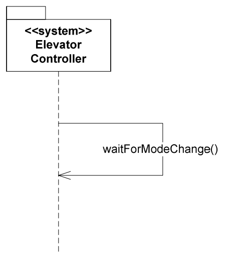

Figure 12: UC7: Suspended in SERVICE mode

3.1.1.2 System State Diagrams – Level 0

The following state diagrams specify EC system states and transitions *at the use case abstraction level*. Composite states are decomposed and fleshed-out in subsequent sub-sections.

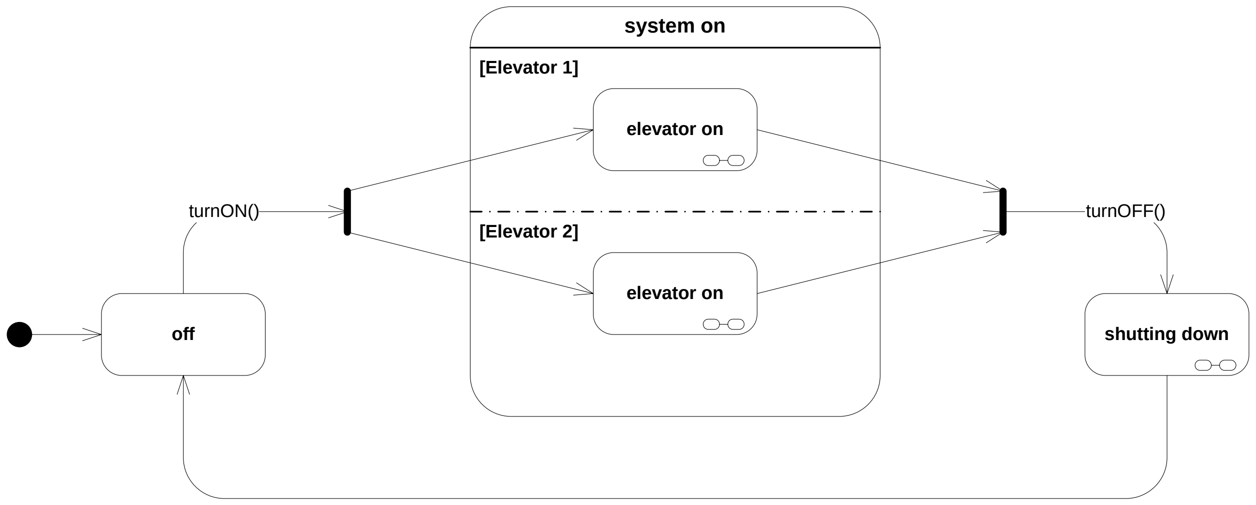

Figure 13: Top-level System State Diagram (Level 0).

Figure 13 illustrates the top-level system states (ON and OFF states) and transitions. At this top-level abstraction, the concurrent management/control of *both* elevator cars (elevator 1 and elevator 2) by a *single* EC instance is captured. All subsequent lower-level diagrams depict only a single instance of the '*elevator on'* composite state (i.e. the control of only a single elevator car). Always keep in mind that, in actuality, two elevator cars are being controlled concurrently (by a single EC instance).

3.1.1.3 System State Diagrams – Level 1

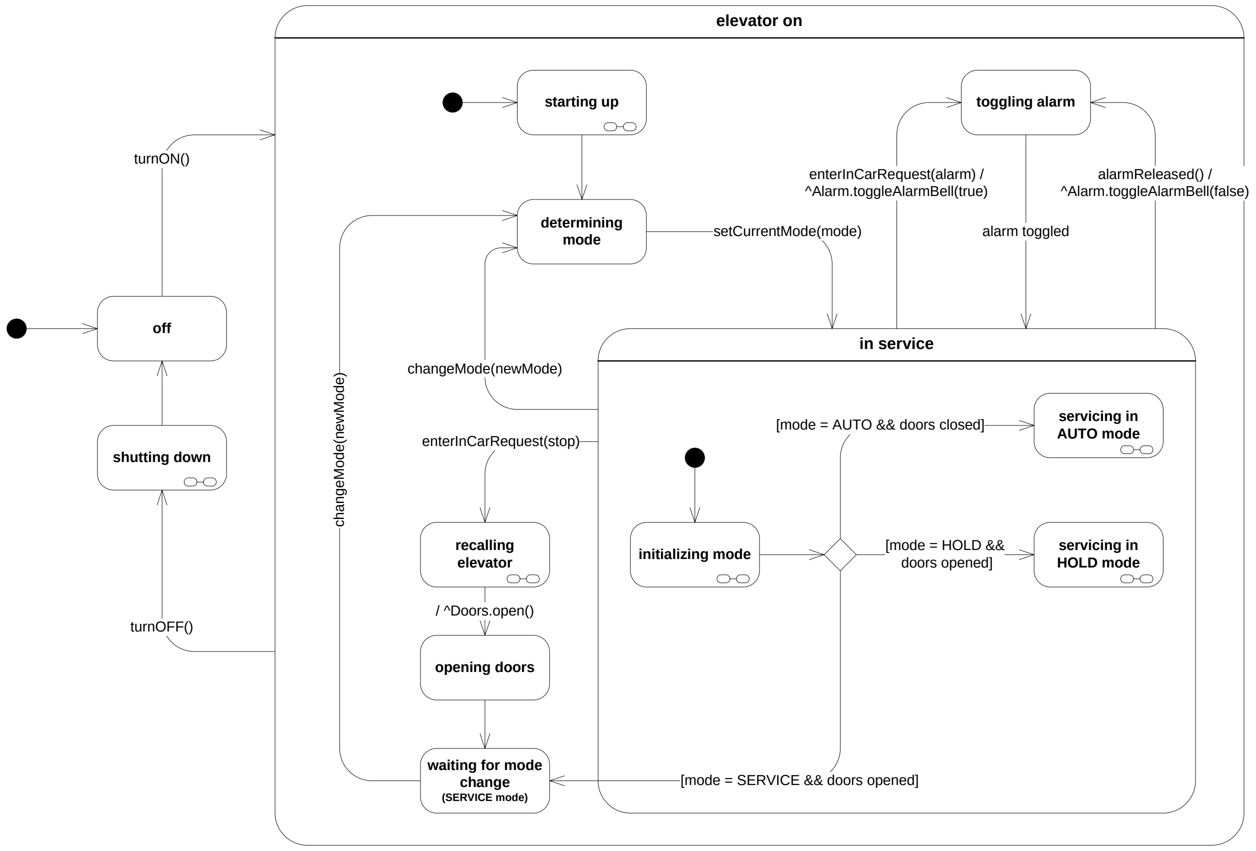

Figure 14: Decomposition of '*elevator on'* state (Level 1).

Note: Recall that there are two concurrent instances of the '*elevator on'* state (machine), one for each elevator car. The '*shutting down'* and '*off'* states have been included in Figure 14 to provide greater context.

Note: The '*determining mode'* state represents the EC determining the position of the in-car mode key-switch.

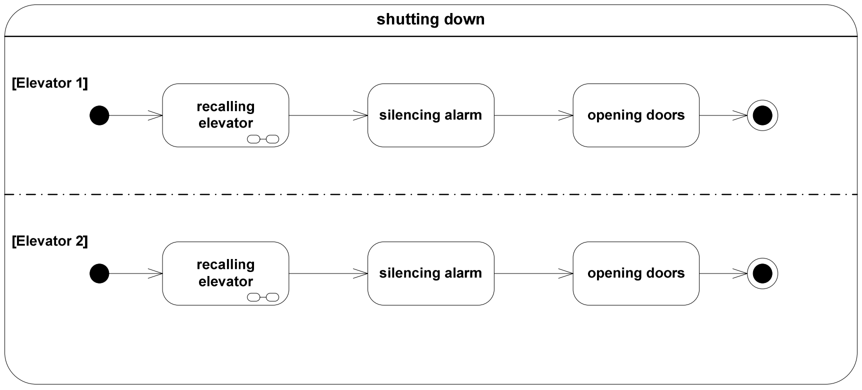

Figure 15: Decomposition of *'shutting down'* state (Level 1).

3.1.1.4 System State Diagrams – Level 2

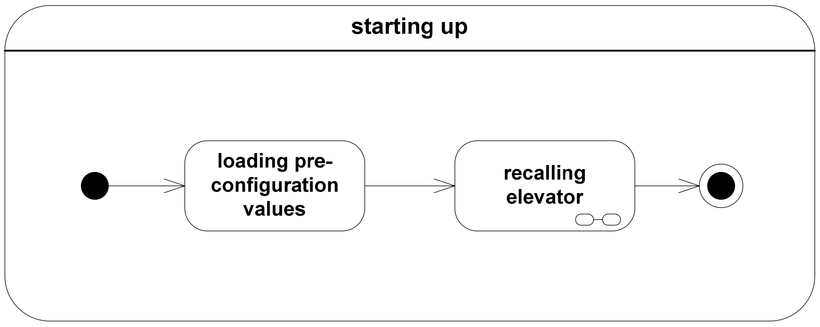

Figure 16: Decomposition of *‘starting up’* state (Level 2).

Note: The “pre-configuration values” referred to if Figure 16 include default/recall floor, maximum elevator capacity (weight), and timeout period (wait time) for allowing passengers to board/exit an elevator car.

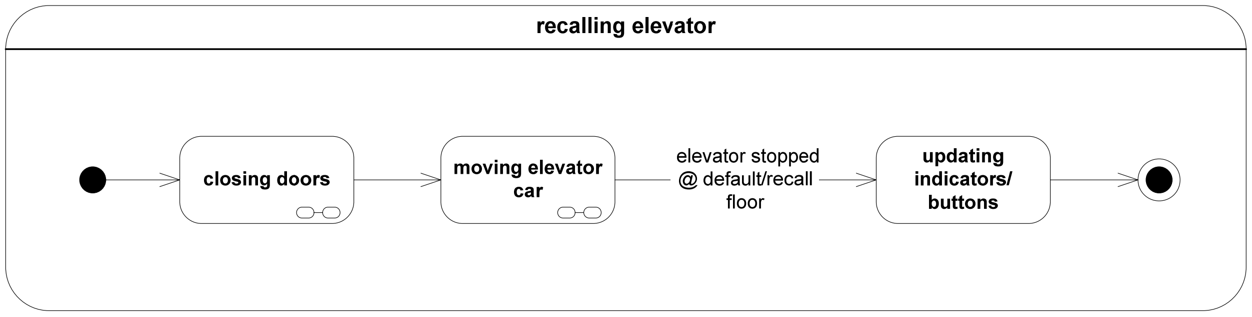
recalling elevator
closing doors
moving elevator car
elevator stopped @ default/recall floor
updating indicators/ buttons

Figure 17: Decomposition of *‘recalling elevator’* state (Level 2).

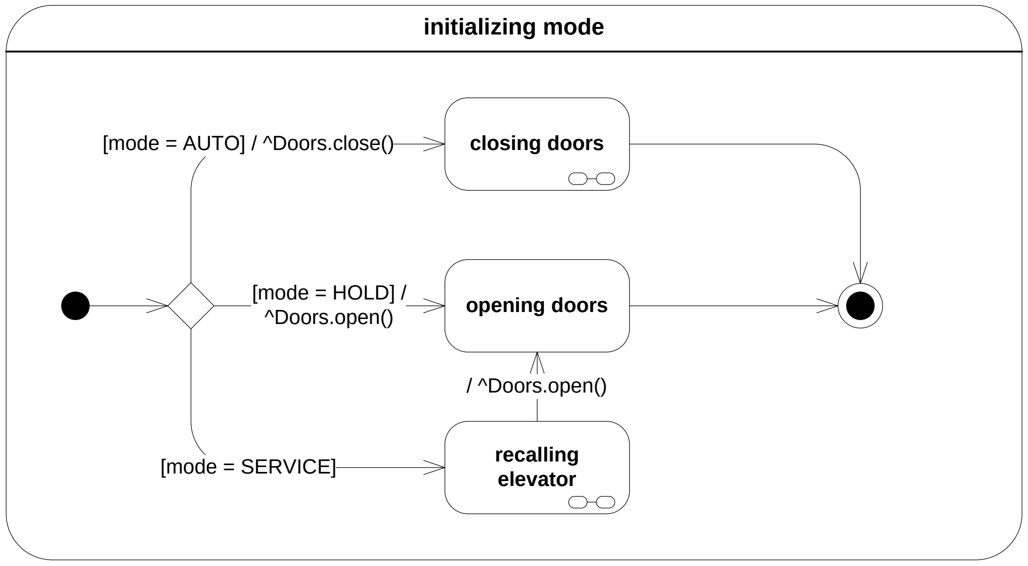

Figure 18: Decomposition of *‘initializing mode’* state (Level 2).

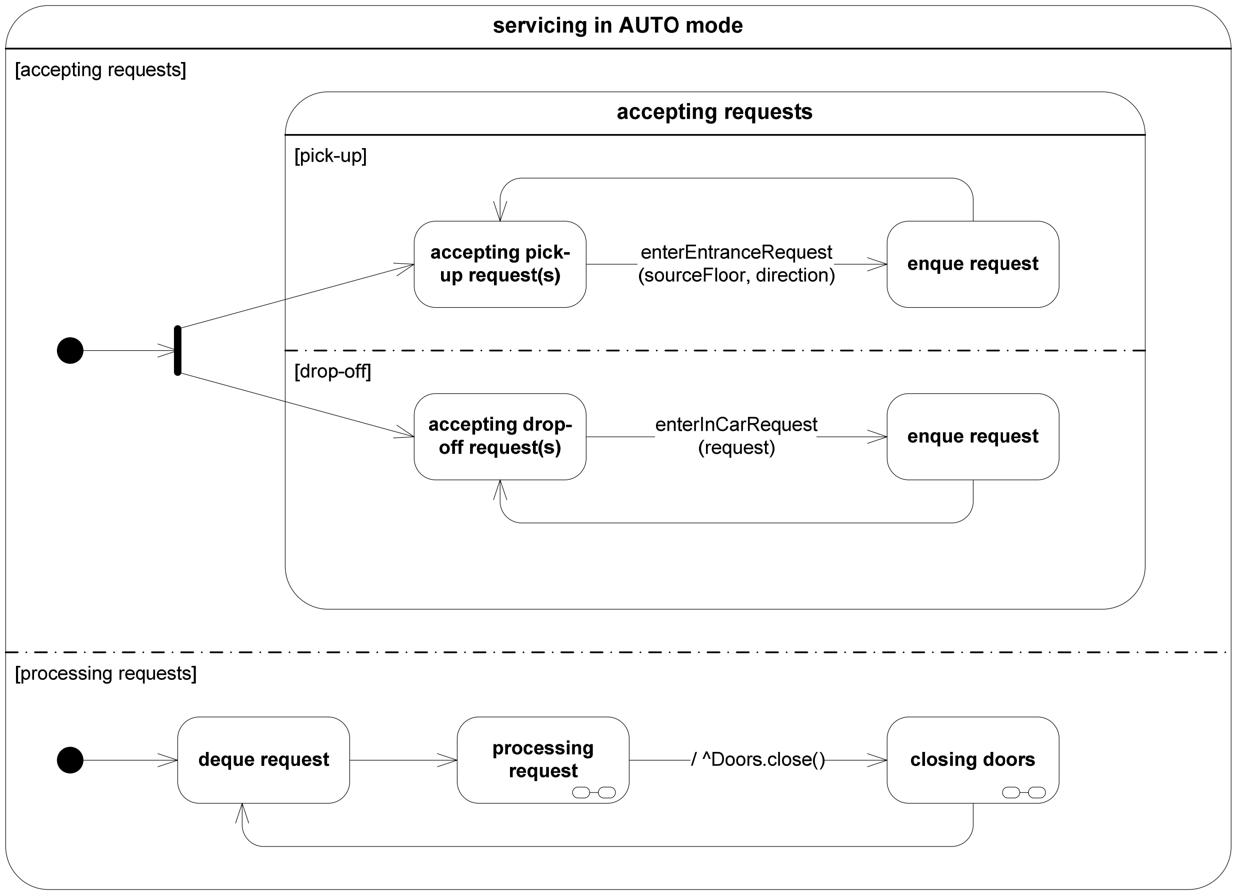

Figure 19: Decomposition of '*servicing in AUTO mode*' state (Level 2).

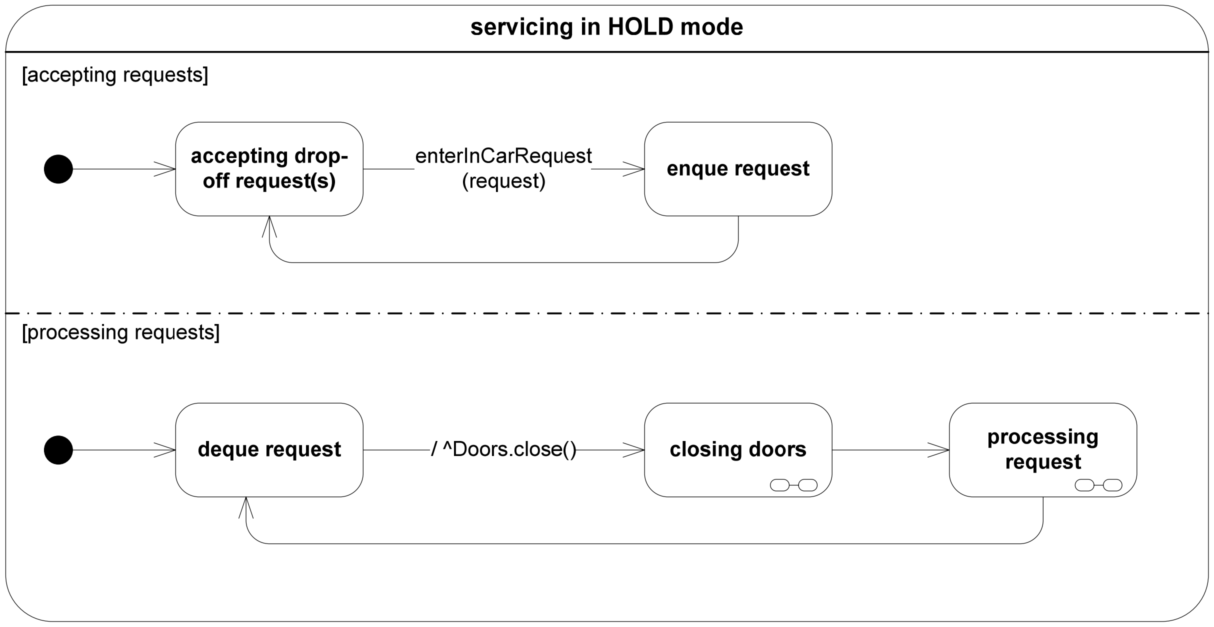

Figure 20: Decomposition of '*servicing in HOLD mode*' state (Level 2).

Note: Figures 19 and 20 describe system states and transitions while operating in AUTO mode and HOLD mode, respectively. Notice the concurrent sub-states allowing the EC to accept subsequent pick-up/drop-off requests while servicing (processing) a request. Requests are intuitively placed in a queue and processed in FIFO order. The physical design/details of this queue should be considered in the software design specification.

Note: The value of *request* (as returned by enterInCarRequest() event) is typically a floor selection, but can also be 'alarm' (to sound the alarm bell), 'stop' (to invoke emergency stop), 'mode' (to signify mode change), etc. Different values result in different state transitions. The range of values for *request* is specified in Section 3.2.2. As well, other state diagrams (figures 14 and 22 in particular) specify how the EC responds to non-floor selection values – take note of these.

3.1.1.5 System State Diagrams – Level 3

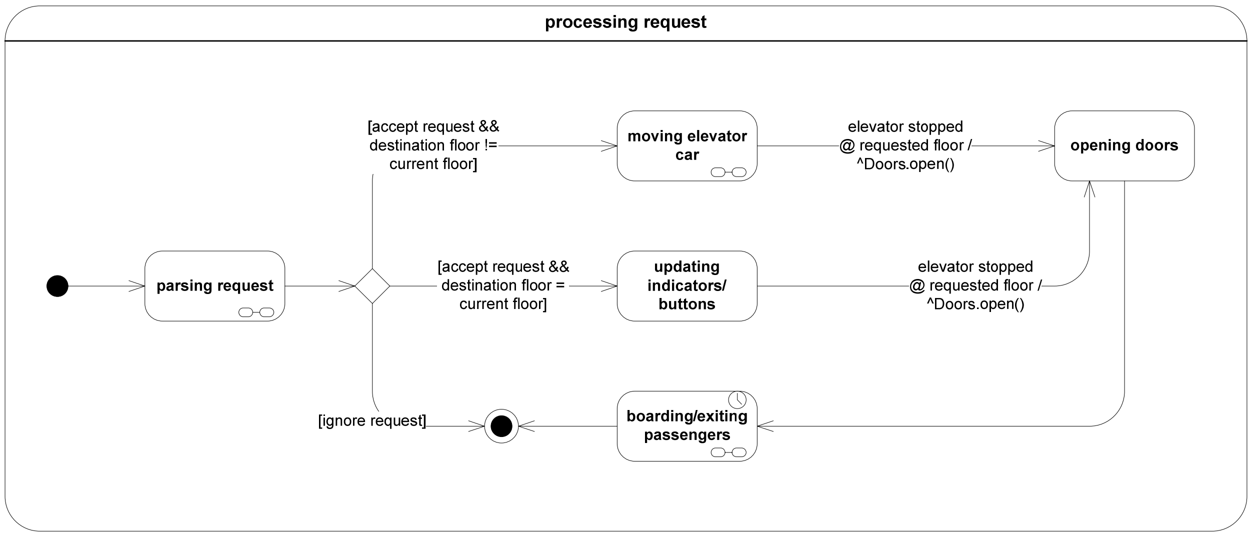
processing request
parsing request
[accept request && destination floor != current floor]
moving elevator car
elevator stopped @ requested floor / ^Doors.open()
opening doors
[accept request && destination floor = current floor]
updating indicators/ buttons
elevator stopped @ requested floor / ^Doors.open()
[ignore request]
boarding/exiting passengers

Figure 21: Decomposition of ‘*processing request’* state (Level 3).

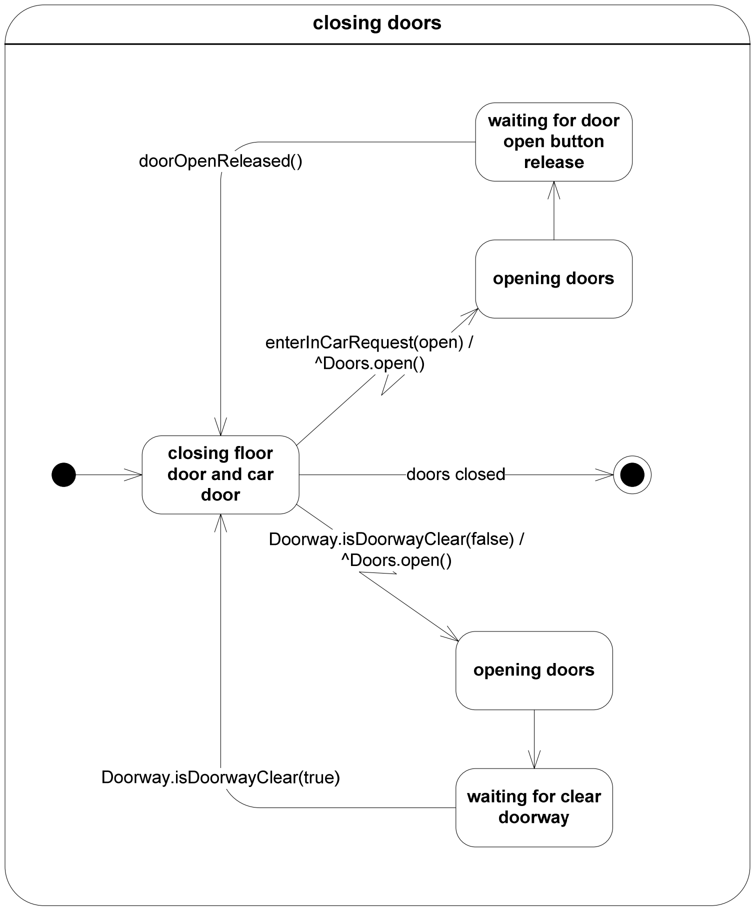
closing doors
waiting for door open button release
doorOpenReleased()
opening doors
enterInCarRequest(open) / ^Doors.open()
closing floor door and car door
doors closed
Doorway.isDoorwayClear(false) / ^Doors.open()
opening doors
waiting for clear doorway
Doorway.isDoorwayClear(true)

Figure 22: Decomposition of *‘closing doors’* state (Level 3).

3.1.1.6 System State Diagrams – Level 4

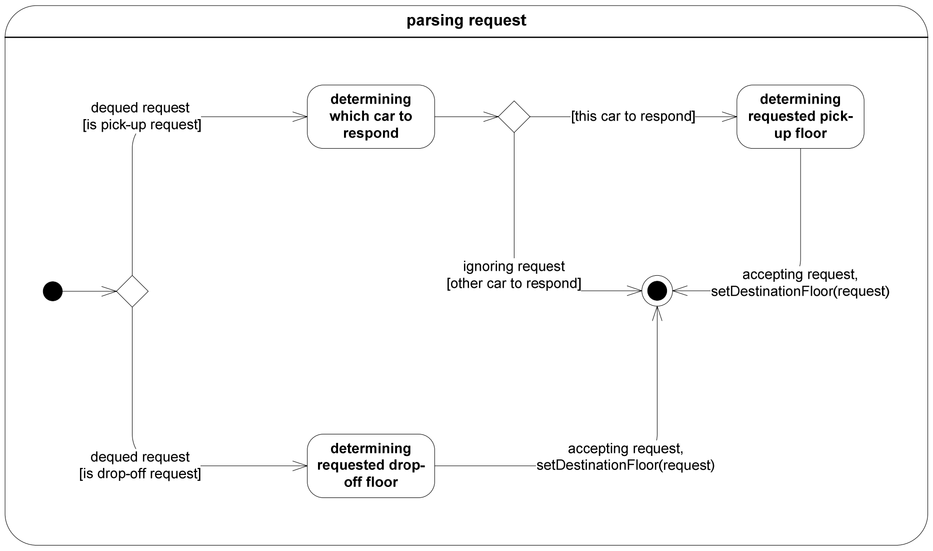

Figure 23: Decomposition of *'parsing request'* state (Level 4).

Note: When the '*parsing request'* state is executing in the context of the '*servicing in HOLD mode'* state, only the bottom conditional branch ([is drop-off request]) is ever invoked. This is correct as pick-up requests are to be ignored while an elevator is operating in HOLD mode.

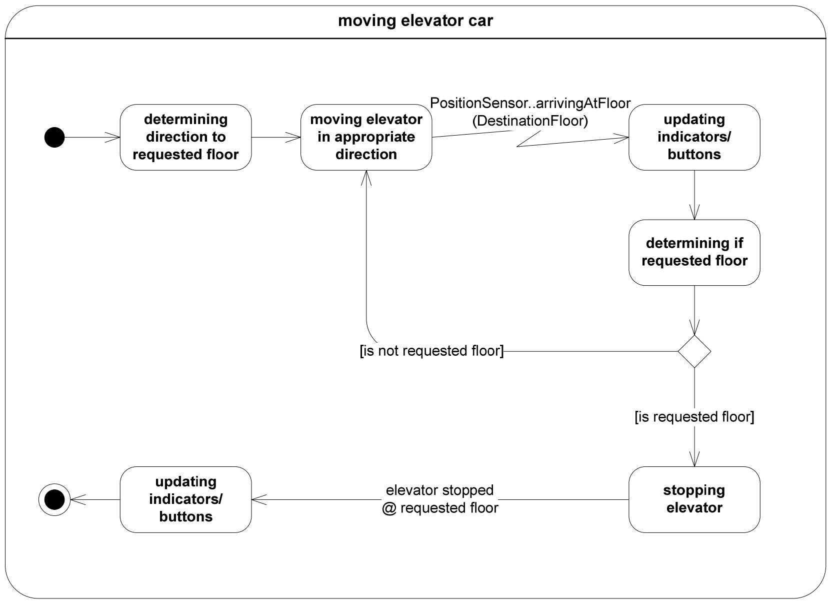
moving elevator car
determining direction to requested floor
moving elevator in appropriate direction
PositionSensor..arrivingAtFloor
(DestinationFloor)
updating indicators/ buttons
determining if requested floor
[is not requested floor]
[is requested floor]
stopping elevator
elevator stopped
@ requested floor
updating indicators/ buttons

Figure 24: Decomposition of *'moving elevator car'* state (Level 4).

Note: The direction to move the elevator car is determined using this simple formula:

```
    if (DestinationFloor > CurrentFloor) { direction := UP }
    else { direction := DOWN }
```

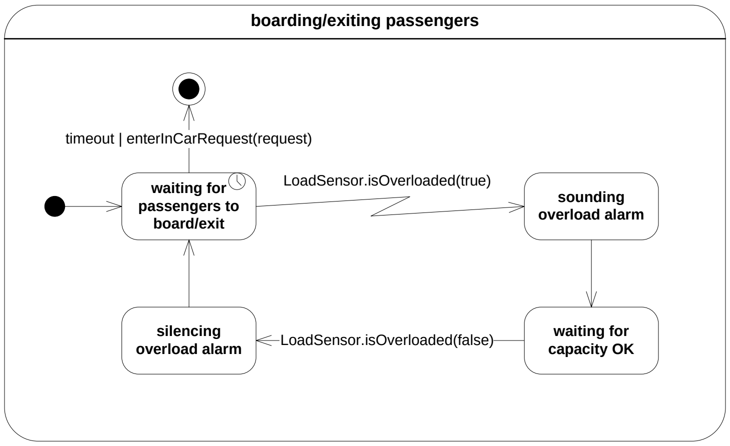
boarding/exiting passengers
timeout | enterInCarRequest(request)
waiting for passengers to board/exit
LoadSensor.isOverloaded(true)
sounding overload alarm
silencing overload alarm
LoadSensor.isOverloaded(false)
waiting for capacity OK

Figure 25: Decomposition of *'boarding/exiting passengers'* state (Level 4).

Note: The *'boarding/exiting passengers'* state is a timed state where transition to the next state occurs automatically after some (pre-configured) timeout period (without any other transitioning events occurring first). When the *'boarding/exiting passengers'* state is executing in the context of the '*servicing in HOLD mode'* state, the timeout period is assumed to be infinite (i.e., this state never times out while in HOLD mode).

3.1.1.7 System Conceptual Diagram

The System Conceptual Diagram specifies the system concepts extracted and derived from system state diagrams. This system-level conceptual diagram also indicates high-level interactions between concepts. Detailed specification of interactions between concepts is provided in Section 3.1.3 Collaboration Diagrams.

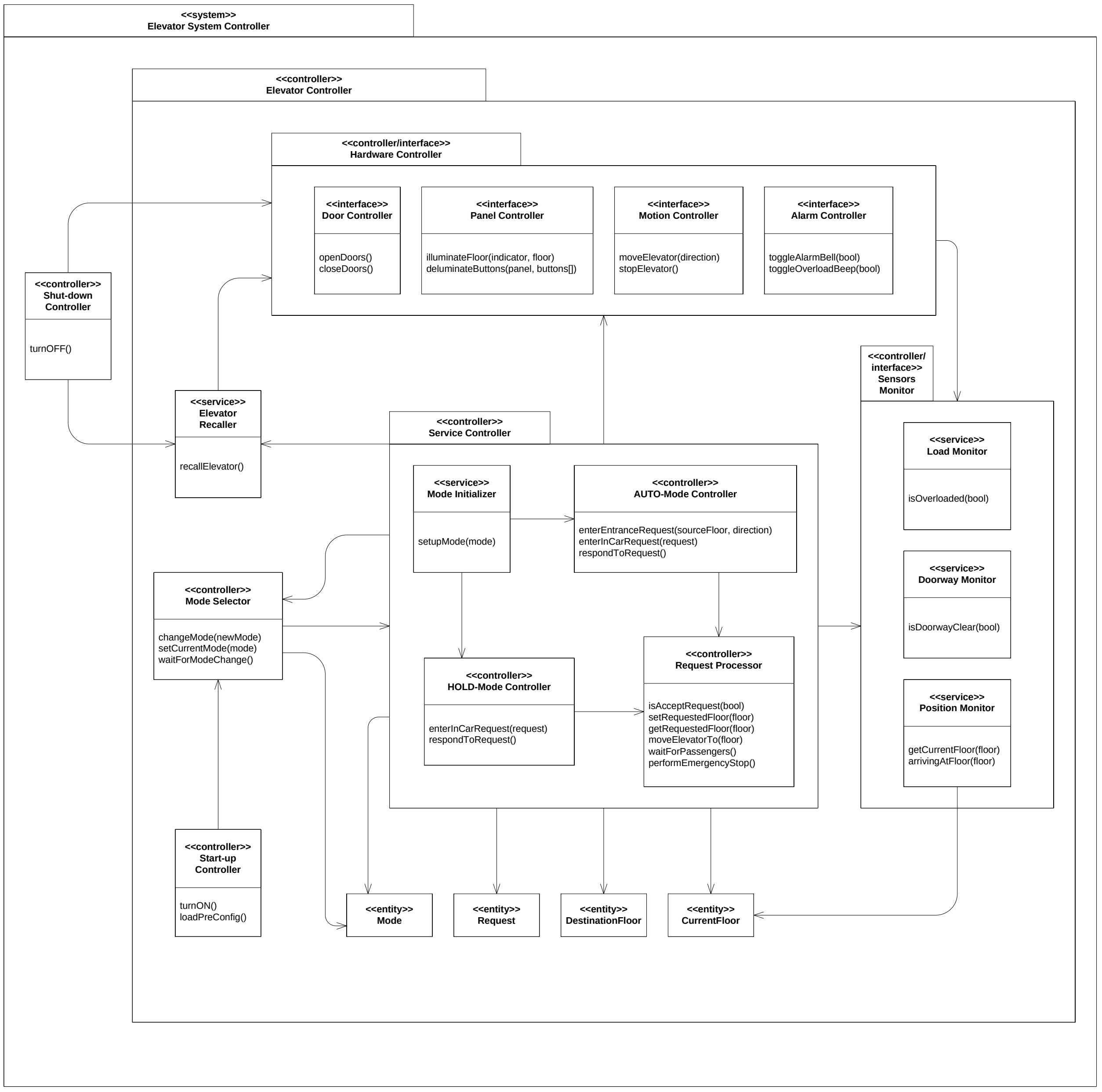

Figure 26: System Conceptual Diagram.

3.1.2 Concept State Diagrams

The following state diagrams specify internal states and transitions of controller objects defined in the System Conceptual Diagram. Internal states and transitions of service and interface objects can be intuitively derived from the System Concept Diagram and system state diagrams – this is trivial for these objects and so explicit state diagrams have been omitted.

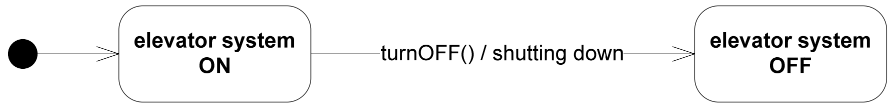

Figure 27: State diagram for Shut-down Controller.

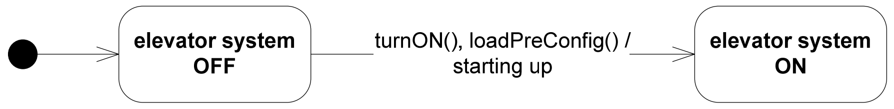

Figure 28: State diagram for Start-up Controller.

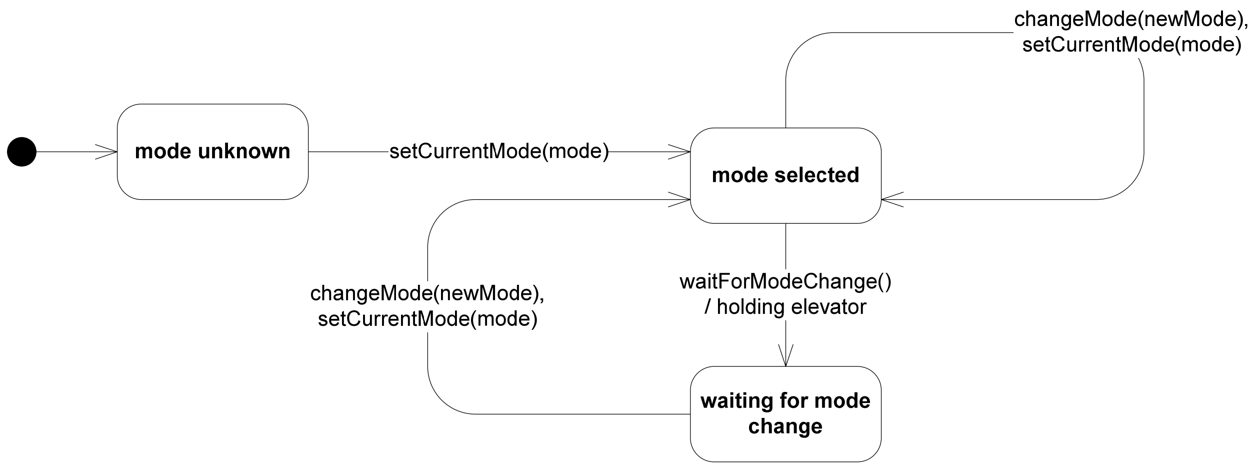
mode unknown
setCurrentMode(mode)
mode selected
changeMode(newMode), setCurrentMode(mode)
waitForModeChange() / holding elevator
waiting for mode change
changeMode(newMode), setCurrentMode(mode)

Figure 29: State diagram for Mode Selector.

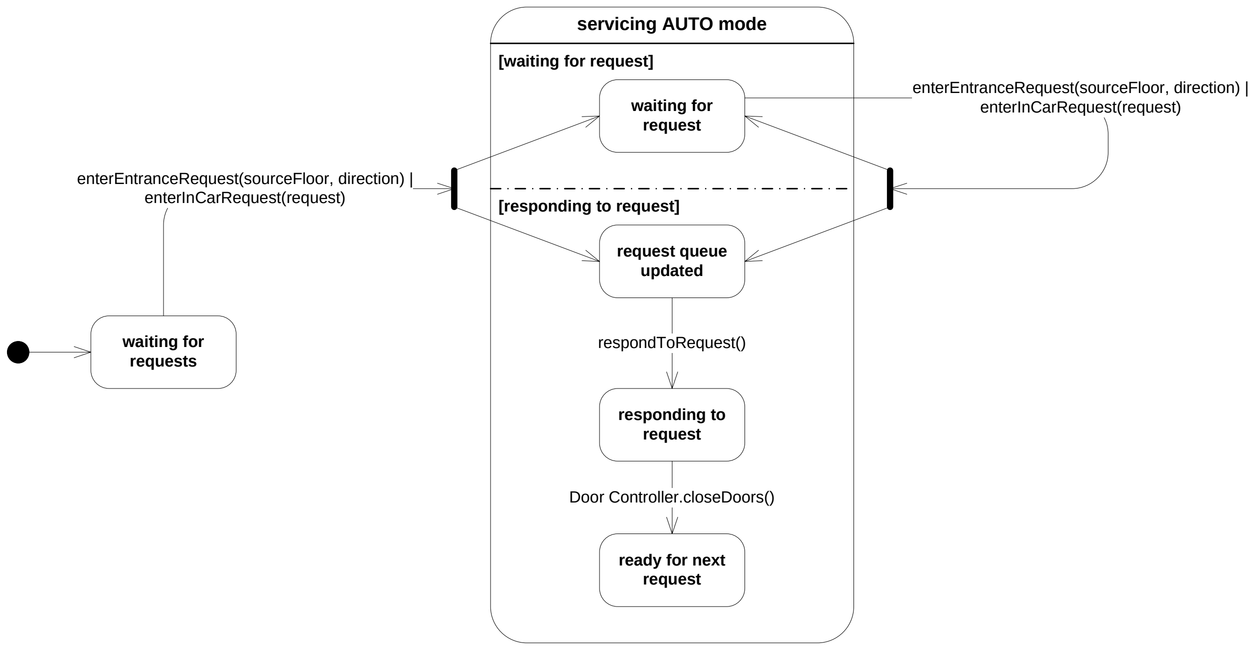

Figure 30: State diagram for AUTO-Mode Controller.

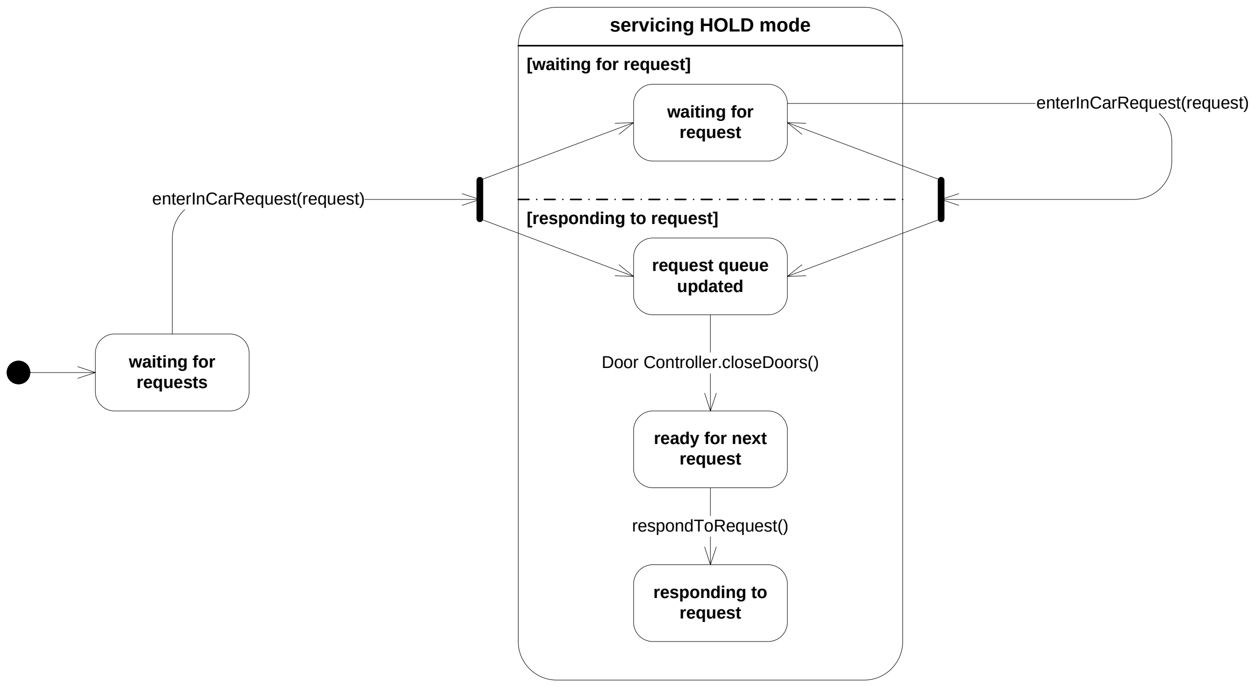

Figure 31: State diagram for HOLD-Mode Controller.

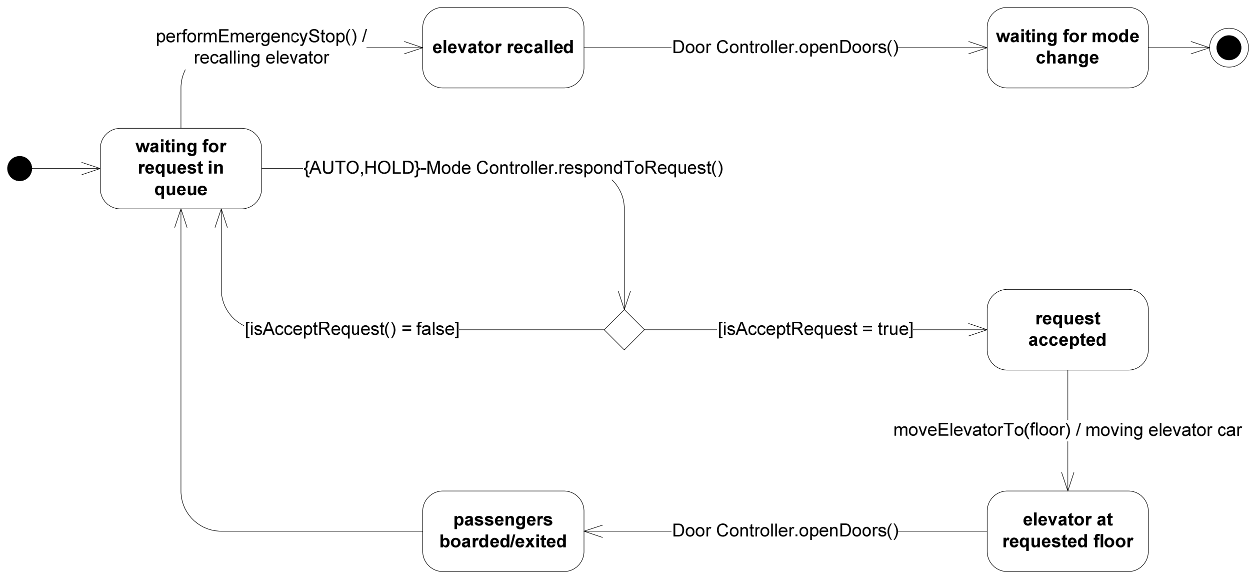
performEmergencyStop() / recalling elevator
elevator recalled
Door Controller.openDoors()
waiting for mode change
waiting for request in queue
{AUTO,HOLD}-Mode Controller.respondToRequest()
[isAcceptRequest() = false]
[isAcceptRequest = true]
request accepted
moveElevatorTo(floor) / moving elevator car
passengers boarded/exited
Door Controller.openDoors()
elevator at requested floor

Figure 32: State diagram for Request Processor.

Note: The `isAcceptRequest()` determines whether the specific elevator car should service a request. If the request came from an in-car button panel selection, then `isAcceptRequest()` returns true, of course. However, in the case where the request came from an up/down button panel (at an elevator entrance), the function should return a value based on the elevator scheduling heuristics. That is, if the *other* elevator car is in AUTO mode AND if it is closer to the requested pick-up floor AND it is already traveling in the correct direction, then return false (i.e. ignore the request and let the other car handle it). Otherwise, return true (i.e. handle the request with this car).

3.1.3 Collaboration Diagrams

The following sections specify interaction between objects (concepts) using collaboration diagrams.

3.1.3.1 Unified Collaboration Diagram

The Unified Collaboration Diagram is shown in two versions for clarity: one version showing internal interactions only, another version showing interactions with external actors.

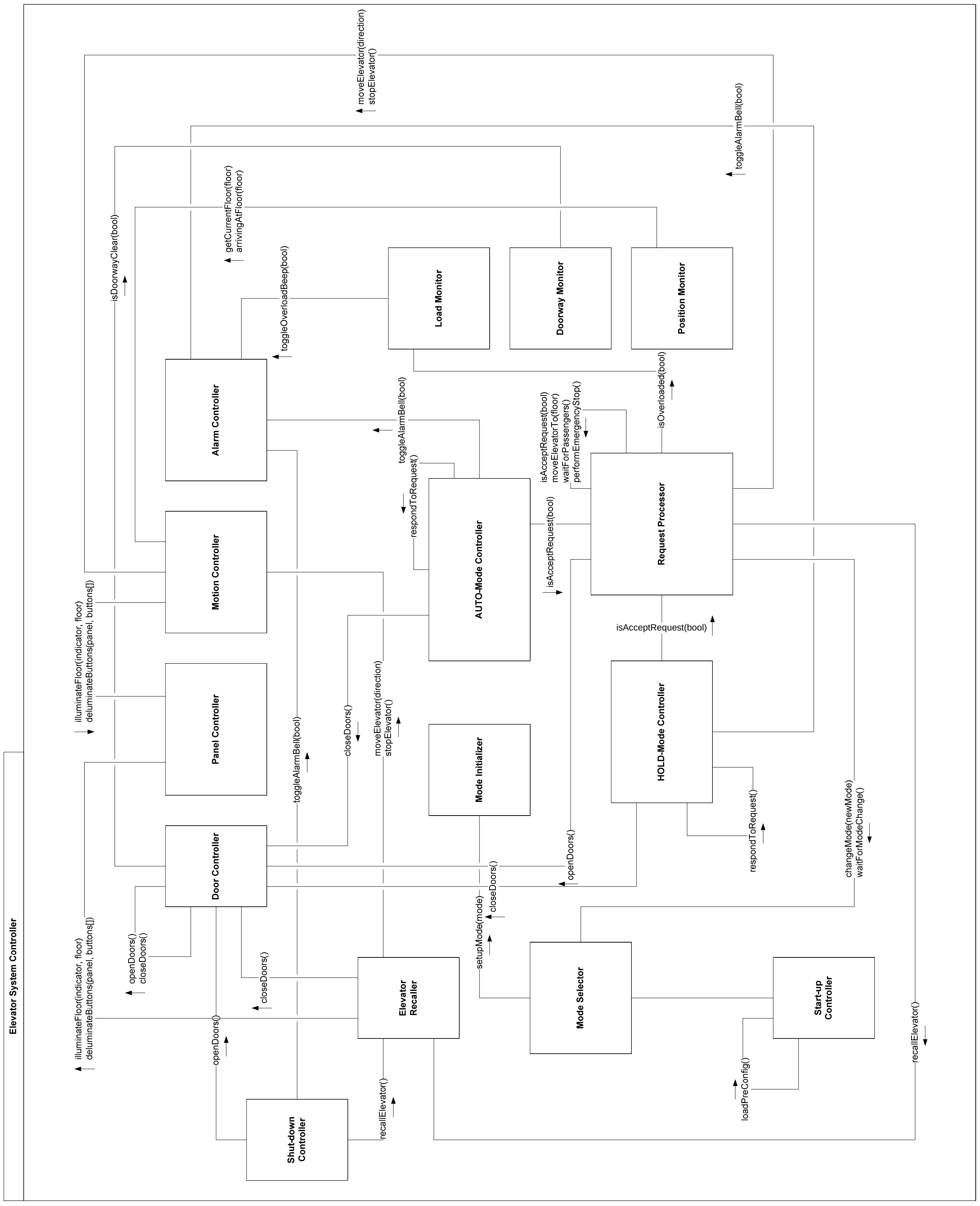

Figure 33: Unified Collaboration Diagram (internal interactions only).

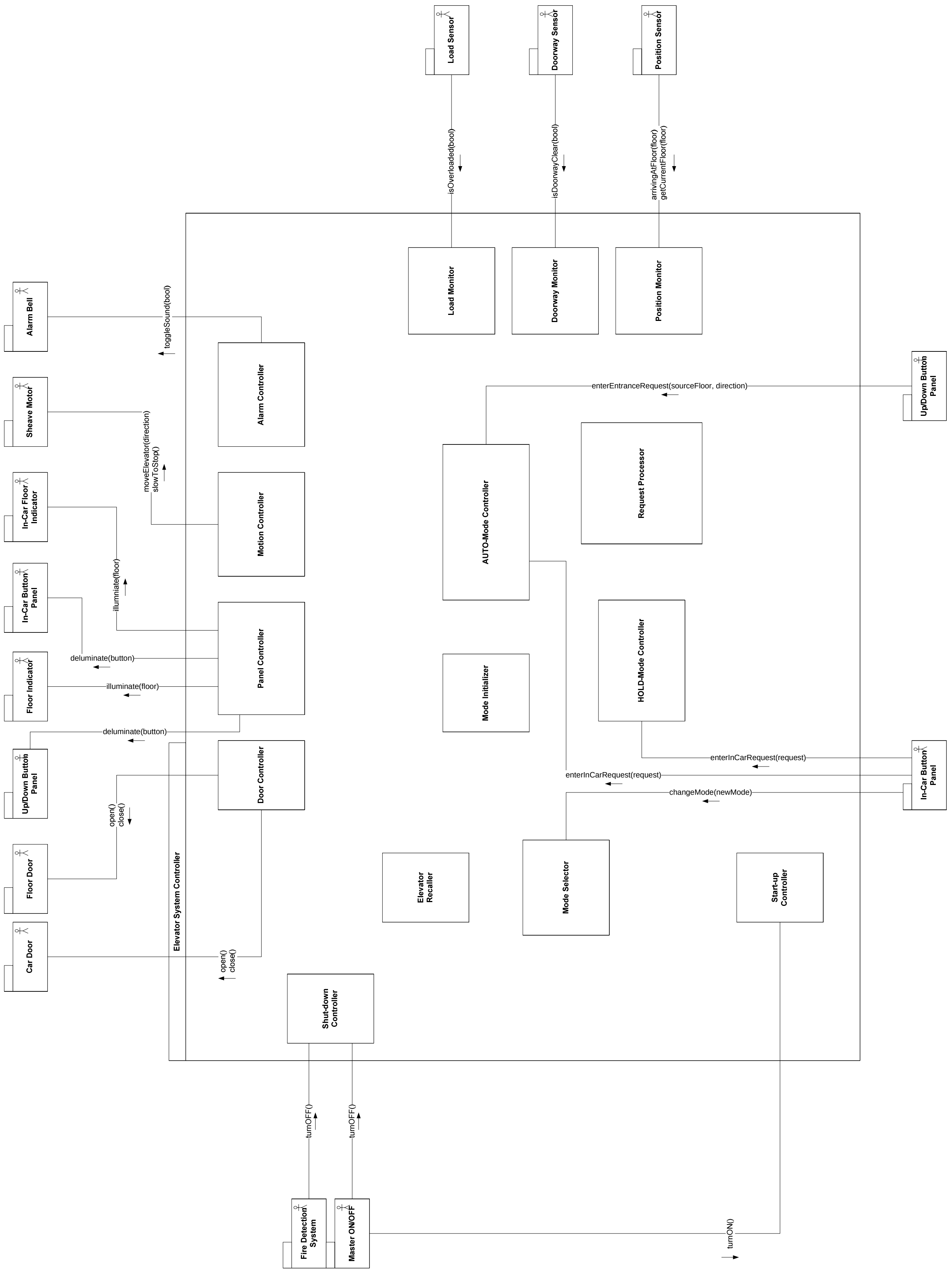

Figure 34: Unified Collaboration Diagram (interactions with external actors).

3.1.3.2 Use Case Collaboration Diagrams

The following collaboration diagrams illustrate object interaction and sequence with respect to AUTO mode use cases (remaining use cases derive relatively trivial collaboration diagrams and so have been omitted).

Note: Some activities from the use case narrations have been decomposed into multiple (lower-level) activities appearing in the collaboration diagram. This is due to the difference in abstraction level between the use case narrations and the system concepts (as noted in Section 4.3). Such decompositions are denoted by the use of .x steps (e.g. 2.1, 2.2, etc.).

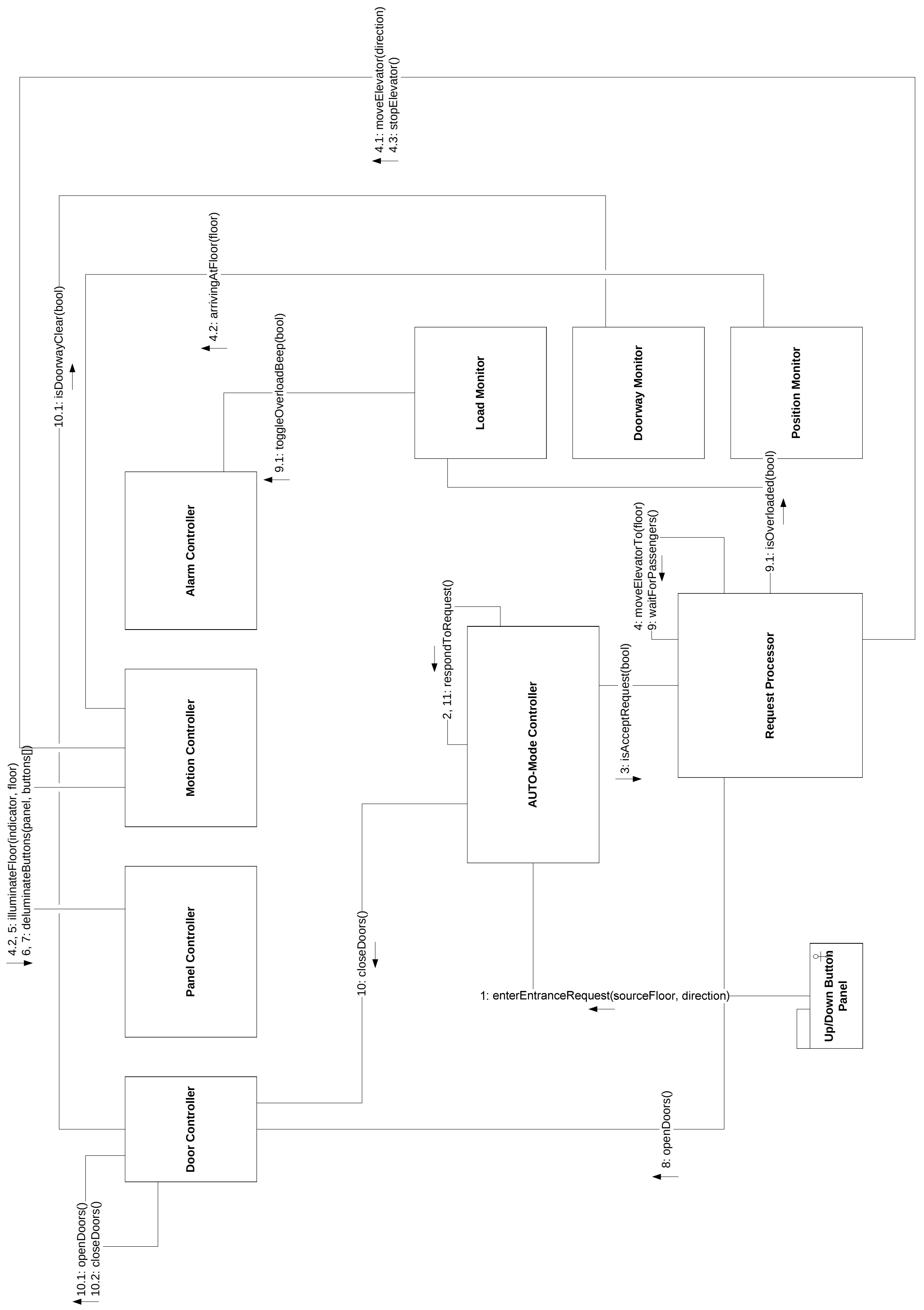

Figure 35: UC4 Collaboration: Pick-Up in AUTO mode

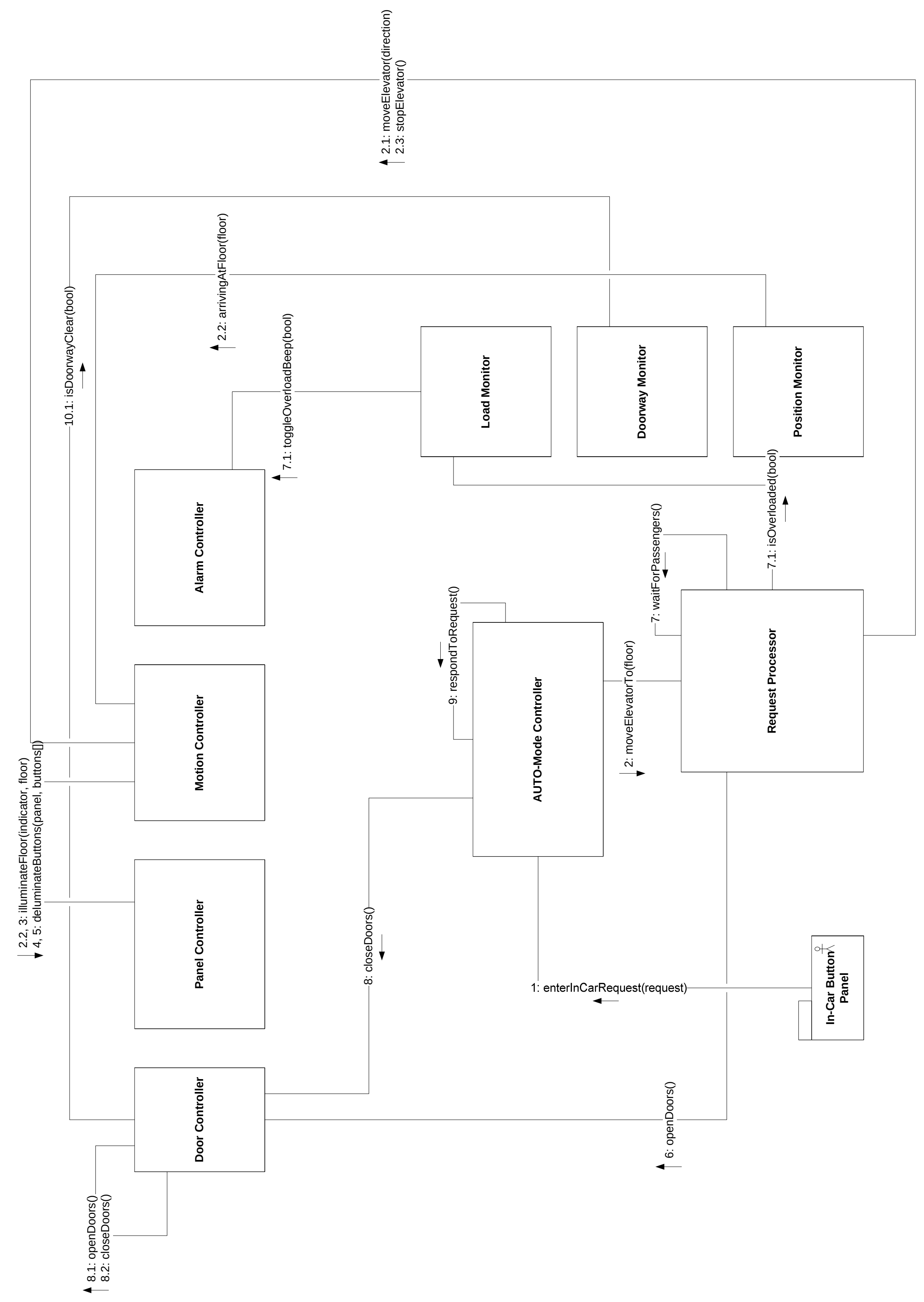

Figure 36: UC5 Collaboration: Drop-Off in AUTO mode

3.2 External Interface Requirements

3.2.1 User Interfaces

The EC does not have an explicit user interface (beyond the buttons and indicators existing as elevator hardware components) and therefore has no specific requirements here.

3.2.2 Hardware Interface – Application Program Interface

As stated in the first assumption listed in Section 2.5.1, a logical abstraction of physical (electrical) signals sent between the elevator controller device and other elevator components is assumed to be available and compatible with the controller device being programmed. For all intents and purposes of this SRS, it is assumed that the abstraction allows for logical naming of well-defined events and function calls, with discrete and well-defined parameters.

The following table lists these logical events and function calls as associated with each elevator component. This table essentially forms the (logical) hardware interface requirements for the EC.

Notation: Logical signal names beginning with underscore (i.e. _<signal name>) describe *event* signals. Logical names without a preceding underscore (i.e. <signal name>) describe *function call* signals. Notice that for events, the EC is the event listener, and for function calls, the EC is the caller. The only exception is turning the EC on and off, which are function calls (commands, rather) that the EC must accept, and where the master switch is the caller.

Elevator Component	Logical Signal Name/Signature	Description
Elevator Controller	turnON()	Turn on the elevator system, invoke the start-up sequence.
	turnOFF()	Turn off the elevator system, invoke the shut-down sequence.
Floor Indicator	illuminate([in] floor)	Illuminate the floor number specified by *floor* (deluminate all other floor numbers). *floor* is a value in {1, 2, 3, 4, …, *n*}
Up/Down Button Panel	_enterEntranceRequest (sourceFloor, direction)	Request entered (from elevator entrance) event, with *direction* indicating the button being pressed (direction being requested) and *sourceFloor* indicating which floor's up/down button panel fired the event . This event is fired when a button is pressed. *sourceFloor* is a value in {1, 2, 3, 4, …, *n*} *direction* is a value in {up, down}
	deluminate([in] button)	Deluminate the button specified by *button*. *Button* is a value in {up, down}
Floor Door	open()	Open the elevator entrance floor door.
	close()	Close the elevator entrance floor door.

In-Car Button Panel	_changeMode(newMode)	Mode change event, with the new mode indicated by *newMode.* This event is fired when the mode key-switch is turned to a new position. *newMode* is a value in {AUTO, HOLD, SERVICE}
	_enterInCarRequest(request)	Request entered event, with *request* indicating the button being pressed (floor/action being requested). This event is fired when a button is pressed or the mode key-switch is turned to a new position. *request* is a value in {1, 2, 3, 4, ..., *n*, open, alarm, stop, mode}
	_doorOpenReleased()	Door open button released event. This event is fired whenever the door open button is released after being pressed (and held).
	_alarmReleased()	Alarm button released event. This event is fired whenever the alarm button is released after being pressed (and held).
	deluminate([in] button)	Deluminate the button specified by *button*. *Button* is a value in {1, 2, 3, 4, ..., *n*}
In-Car Floor Indicator	illuminate([in] floor)	Illuminate the floor number specified by *floor* (deluminate all other floor numbers). *floor* is a value in {1, 2, 3, 4, ..., *n*}
Sheave Motor	moveElevator([in] direction)	Move the elevator car in the direction specified by *direction.* *direction* is a value in {up, down}
	slowToStop()	Decelerate and stop the elevator car at the nearest floor (relative to the current direction of travel).
Position Sensor	_position(floor)	New floor event, with the (approaching) floor indicated by *floor*. This event is fired whenever the position sensor detects a discrete change in elevator location from one floor to the next. *floor* is a value in {1, 2, 3, 4, ..., *n*}
	position([retval] floor)	Return the current elevator location with *floor* being the return value. *floor* is a value in {1, 2, 3, 4, ..., *n*}
Doorway Sensor	_doorwayClear(bool)	Doorway clearance change event, with new clearance status indicated by *bool*. This event is fired whenever the doorway clearance changes from clear/unblocked (*bool* is true) to blocked (*bool* is false), or vice versa. *bool* is a value in {true, false}

Load Sensor	_overLoad(bool)	Overload status change event, with new status indicated by *bool*. This event is fired whenever the elevator load changes from within load limits (*bool* is false) to above load limits, i.e. overloaded (*bool* is true), or vice versa. *bool* is a value in {true, false}
Car Door	open()	Open the elevator car door.
	close()	Close the elevator car door.
Alarm Bell	toggleSound(bool)	Toggle the alarm bell as specified by *bool* (true for sounding, false for silenced). *bool* is a value in {true, false}

Table 2: Hardware interface requirements.

Note: The signals tabulated above indeed synchronize, in general, with the signals and commands depicted in the EC context diagram in Section 2.1.

3.2.3 Communications Interfaces

The EC does not utilize any communications mechanisms (beyond direct signaling between elevator components) and therefore has no specific requirements here.

4 Reference Tables & Descriptions

4.1 Functional Requirements Table & Traceability Information

ID	Name	Description	Details/Constraints	Category	Related Req'ts	Source (§1.4)	Related Use Cases (§4.3)	Where Specified
F1	Up/Down Movement	Control up/down motion of elevator cars.	Stop elevator cars at floors as requested by passengers.	E	F2, F3	[1], [4]	UC4, UC5, UC6	§3.1
F2	Open/Close Doors	Control opening/closing entrance doors and elevator car doors.	Allow entrance/exit of passenger to/from elevator car. As a safety measure, close doors only when doorway is clear.	E	F1, F3, F8, F11	[1], [4]	UC4, UC5, UC6	§3.1
F3	Modes of Operation/ Recall	Support two operational modes and one recall mode.	Three modes: AUTO, HOLD, SERVICE. See §2.2 for functional description of each mode.	E	F1	[3], [4]	UC3, UC4, UC5, UC6, UC7	§2.2, §3.1

F4	Floor Indicators	Update floor indicators with elevator car positions.	Panels located in elevator cars and above each floor entrance.	E	F5	[1], [4]	UC4, UC5, UC6	§3.1
F5	Button Illumination	Directional and floor buttons illuminate when pressed. EC must deluminate these buttons.	Buttons should be deluminated once elevator request has been fulfilled (e.g.arrived at destination or pick-up floor)	E	F4	[1], [4]	UC4, UC5, UC6	§3.1
F6	Start Up State	Both elevators should positioned be returned to the (pre-configured) default/recall floor at system start-up.	State of doors after start-up depends on the current mode of operation.	I	F3, F7	[4]	UC1	§3.1
F7	Shut Down State	Both elevators should be returned to the (pre-configured) default/recall floor at system shut-down.	Default/recall floor doors and elevator car doors should be in open states after shut-down.	I	F6, F13	[4]	UC2	§3.1
F8	Door-Open Override	The door-open button, when pressed-and-held will override any attempt to close the door.	The button signal is only used when doors are already open. Once doors are fully closed, the door-open button is ignored.	E	F2	[4]	UC4, UC5, UC6	§3.1
F9	In-car Safety Alarm	An in-car alarm can be triggered at any time by pressing and holding the alarm button. The elevator functionality shall proceed as usual even with alarm sounding. The purpose of the alarm is to gain attention either for passengers of the particular elevator car, or for the car itself.	The alarm is not an indication of malfunction, thus, the elevator can proceed to be used even with alarm engaged. The alarm is silenced as soon as the alarm button is released.	E	F10	[3], [4]	UC4, UC5, UC6	§3.1

F10	Emergency Stopping	The emergency stop button shall be used to return the elevator car to the (pre-configured) default/recall floor immediately. It can be triggered at any time by pressing the emergency stop button.	When this button is pressed, the elevator car is to go to a (pre-configured) default/recall floor and open the elevator doors. Reanimation of the stopped elevator car then requires operator action (operator must use the in-car mode key switch to reset the stopped car by switching to any mode [and back] or restart the entire elevator system from control room).	E	F9	[3], [4]	UC4, UC5, UC6	§2.2
F11	Overload Detection	An elevator must never close its doors when the load it bears is greater than [pre-configured] max. capacity.	A load sensor provides load information.	E	F2	[4]	UC4, UC5, UC6	§3.1
F12	Idle Behaviour	An elevator car should simply stay on the most recent floor it was on when waiting for new requests.	No need to re-position elevator during idle.	I		[4]		§3.1
F13	Fire Detection	The building's Fire Detection System should be able to control the elevator system in the event of a fire.	If a fire is detected, the Fire Detection System can force a shut-down of the elevator system (thus initiating the usual shut-down sequence).	E	F7	[3], [4]	UC2	§3.1

Category: E = explicit, I = implicit, O = optional

Table 3: Functional requirements table.

4.2 Non-Functional Requirements & Traceability Information

ID	Name	Description	Details / Constraints	Category	Related Req'ts	Source (§1.4)	Related Use Cases (§4.3)	Where Specified
N1	Hardware Compatibility	EC must support existing hardware components.	Sheave motor, button panels, floor indicators, sensors, etc.	M		[2], [3]		§2.1, §3.2
N2	Existing Usage Compatibility	EC must provide an elevator experience that is similar to what is already known by typical elevator users.	E.g. support for the 3 modes, functionality of each button, etc.	M	F1, F2, F3, F4, F5	[2], [3], [4]	UC4, UC5, UC6	§2.4
N3	Operator Rights & Security	Operational actions are protected by physical key-switches.	I.e. turning the elevator system on/off, changing elevator mode.	M	F3, F6, F7	[4]	UC1, UC2, UC3	
N4	Elevator Scheduling	When a pick-up request is made, the closest elevator should be sent to fulfill the request.	I.e. the closest elevator already traveling in the right direction (or not traveling at all)	W	F1	[4]	UC4	§2.5, §3.1
N5	Pre-configuration	The EC expects some parameters to be pre-configured (outside of EC's scope). The EC treats these parameters as read-only constants.	These parameters are: - Default/Recall floor - Maximum elevator capacity (weight) - Timeout period (wait time) before (attempting to) close elevator doors	M	F2, F3, F6, F7, F10, F11	[3]	UC1, UC2, UC3, UC4, UC5, UC6	§3.1
N6	EC Scalability	A single EC instance must control exactly two elevator cars concurrently.	The ability to control more than two elevator cars is a non-requirement.	M		[4]		

N7	EC Availability	The EC must be robust and highly-available.	100% up-time with zero maintenance should be the goal. This requirement must be further considered in the software design specification.	M		[4]		

Category: M = must have, W = wanted, nice-to-have

Table 4: Non-functional requirements table.

4.3 Use Cases

The following sections describe the use cases for the EC. For reasons of clarity and effective abstraction, the use cases are described with end-users (passengers, operators) interfacing directly with the EC. Thus, end-users become actors in these use cases rather than the actual hardware components (e.g. button panels, sheave motor, etc.) and the end-users are depicted as interfacing directly with the EC. It should be obvious that, in actuality, the EC interfaces with the elevator hardware components directly, and with the end-users only indirectly. For example, when a use case states that an end-user pushes an elevator button, the end-user is not communicating with the EC directly, but instead, the button is pushed, and a signal is sent from the button panel to the EC. This indirection is implied and the button panel is not listed explicitly as an actor for the use case. Similarly, when a use case states that the EC moves the elevator from one floor to the next, the EC is not physically moving the elevator car, but instead is signaling the sheave motor and, with input from the position sensor, controls the sheave motor to move the elevator car to a specific floor.

Note: This abstraction *at the use case level* should not cause deterioration of correctness or validity of specification artifacts downstream (i.e. derived from the use cases). The processes used to derive these specification artifacts implicitly force (re-)alignment to the proper abstraction level (where actors are in fact elevator hardware components rather than end-users). The use cases have not been updated to reflect the re-alignment, nor do they need to be.

4.3.1 UC1: Turn elevator system ON

Goal:	Operator turns on elevator system.
Pre-Cond.:	Elevator system is turned off, Operator has access (keys) to control room and master ON/OFF key-switch.
Post-Cond.:	Elevator system is turned on.
Event:	Operator turns on elevator system from control room (master ON/OFF).
System:	Elevator Controller (EC)
Actors:	Operator (Op)
Overview:	Description of EC start-up sequence.
Related UCs:	UC2, UC4, UC5, UC6, UC7

Process Description:

1. [Op] Turn master key-switch in elevator control room to ON position.

2. [EC] Close all elevator doors.
3. [EC] Move both elevators to default/recall floor.
4. [EC] Illuminate default/recall floor on all floor indicators.
5. [EC] Deluminate all buttons.
6. [EC] Determine position of mode key-switch.

Alternatives:

AUTO mode start-up

7. [EC] Wait for signal from up/down button panels or in-car button panel.
8. (Continue with UC4 or UC5.)

HOLD mode start-up

7. [EC] Open default/recall floor door and elevator car door.
8. [EC] Wait for signal from in-car button panel.
9. (Continue with UC6.)

SERVICE mode start-up

7. [EC] Open default/recall floor door and elevator car door.
8. [EC] Wait for in-car mode key-switch position change (i.e. do nothing until elevator mode is switched out of SERVICE).

Exceptions:

System shut-down initiated mid-flight

a) [Op] Turn master key-switch in elevator control room to OFF position.
b) [EC] Perform shut-down sequence as specified in UC2.

Door blockage detected during attempt to close door

a) [EC] Detect door blockage (interrupt by doorway sensor).
b) [EC] Open blocked elevator door.
c) [EC] Wait for unblocking (doorway sensor signal).
d) [EC] Close elevator door.
e) (Continue with process step following door close.)

4.3.2 UC2: Turn elevator system OFF

Goal:	Operator *or* Fire Detection System turns off elevator system.
Pre-Cond.:	Elevator system is turned on, Operator has access (keys) to control room and master ON/OFF key-switch.
Post-Cond.:	Elevator system is turned off.
Event:	Operator turns off elevator system from control room (master ON/OFF).
System:	Elevator Controller (EC)
Actors:	Operator (Op), Fire Detection System [FDS]
Overview:	Description of EC shut-down sequence.
Related UCs:	UC1

Process Description:

1. [Op] Turn master key-switch in elevator control room to OFF position.
2. [EC] Close all elevator doors.

3. [EC] Silence all elevator alarm bells.
4. [EC] Move both elevators to default/recall floor.
5. [EC] Open both default/recall floor doors and both elevator car doors.
6. [EC] Illuminate default/recall floor on all floor indicators.
7. [EC] Deluminate all buttons.
8. [EC] EXIT

Alternatives:

1. [FDS] Force elevator system shut-down.

Exceptions:

Door blockage detected during attempt to close door

a) [EC] Detect door blockage (interrupt by doorway sensor).
b) [EC] Open blocked elevator door.
c) [EC] Wait for unblocking (doorway sensor signal).
d) [EC] Close elevator door.
e) (Continue with process step following door close.)

4.3.3 UC3: Change elevator mode / reset emergency stop

Goal:	Operator changes elevator mode and/or reanimates elevator after emergency stop.
Pre-Cond.:	Elevator system is turned on, Operator has access (keys) to mode key-switch. Elevator doors are open.
Post-Cond.:	Elevator mode changed per mode key-switch position.
Event:	Operator changes the elevator mode for an elevator.
System:	Elevator Controller (EC)
Actors:	Operator (Op)
Overview:	Description of EC behaviour upon mode change for an elevator.
Related UCs:	UC4, UC5, UC6, UC7, UC2

Process Description:

1. [Op] Enter elevator car.
2. [Op] Turn mode key-switch in elevator car.
3. [EC] Close current floor door and elevator car door after brief timeout (allow time for Op to exit elevator car – optional of course)
4. [EC] Determine position of mode key-switch.

Alternatives:

Switched to AUTO mode

5. [EC] Wait for signal from up/down button panels or in-car button panel.
6. (Continue with UC4 or UC5.)

Switched to HOLD mode

5. [EC] Open current floor door and elevator car door.
6. [EC] Wait for signal from in-car button panel.
7. (Continue with UC6.)

Switched to SERVICE mode

5. [EC] Move elevator to default/recall floor.
6. [EC] Illuminate default/recall floor on all floor indicators.
7. [EC] Deluminate all buttons.
8. [EC] Open default/recall floor door and elevator car door.
9. [EC] Wait for in-car mode key-switch position change (i.e. do nothing until elevator mode is switched out of SERVICE).

In-car door-open button pressed-and-held

3. [Op] Press-and-hold door-open button.
4. [EC] Open current floor door and elevator car door
5. [Op] Release door-open button.
6. [EC] Continue with default process step 3, above.

Exceptions:

System shut-down initiated mid-flight

a) [Op] Turn master key-switch in elevator control room to OFF position.
b) [EC] Perform shut-down sequence as specified in UC2.

Door blockage detected during attempt to close door

a) [EC] Detect door blockage (interrupt by doorway sensor).
b) [EC] Open blocked elevator door.
c) [EC] Wait for unblocking (doorway sensor signal).
d) [EC] Close elevator door.
e) (Continue with process step following door close.)

Audible alarm triggered

a) [Pa/Op] Press and hold in-car alarm button.
b) [EC] Sound alarm bell.
c) (Continue as usual, but with alarm on.)
d) [Pa/Op] Release alarm button.
e) [EC] Silence alarm bell.

Emergency stop triggered

a) [Pa/Op] Press in-car emergency stop button.
b) [EC] Close elevator doors.
c) [EC] Move elevator to default/recall floor.
d) [EC] Illuminate default/recall floor on all floor indicators.
e) [EC] Deluminate all buttons.
f) [EC] Open default/recall floor door and elevator car door.
g) [EC] Wait for shut-down (UC2) or mode change (UC3) to reset.

4.3.4 UC4: Use elevator in AUTO mode – Pick-Up

Goal:	Passenger embarks (picked-up by) elevator car.
Pre-Cond.:	Elevator system is turned on and in AUTO mode.
Post-Cond.:	Elevator car (with passenger aboard) awaits destination selection or next

pick-up request, doors are closed. Elevator car is located on floor from which passenger was picked up.

Event: Passenger (at elevator entrance, not yet in elevator car) requests elevator service from building floor.
System: Elevator Controller (EC)
Actors: Passenger (Pa)

Overview: Description of EC behaviour in AUTO mode for passenger pick-up.

Related UCs: UC5, UC3, UC2

Process Description:

1. [Pa] Push a button on up/down button panel at an elevator entrance.
2. [EC] Determine which floor has been requested for pick-up.
3. [EC] Determine which elevator car should be used for pick-up (poor man's optimization: use car that is in AUTO mode, traveling in the correct direction, and is closest to the pick-up floor).
4. [EC] Move selected elevator car to pick-up floor.
5. [EC] Update in-car and corresponding entrance floor indicators at each floor.
6. [EC] Deluminate appropriate directional button on floor up/down button panel.
7. [EC] Deluminate corresponding in-car floor button (may or may not have been illuminated)
8. [EC] Open final floor door and elevator car door.
9. [Pa] Enter elevator car.
10. [EC] Close current floor door and elevator car door after brief timeout.
11. [EC] Wait for signal from up/down button panels or in-car button panel.

Alternatives:

In-car door-open button pressed-and-held

10. [Pa] Press-and-hold door-open button.
11. [EC] Open current floor door and elevator car door
12. [Pa] Release door-open button.
13. [EC] Continue with default process step 10, above.

Exceptions:

System shut-down initiated mid-flight

a) [Op] Turn master key-switch in elevator control room to OFF position.
b) [EC] Perform shut-down sequence as specified in UC2.

Door blockage detected during attempt to close door

a) [EC] Detect door blockage (interrupt by doorway sensor).
b) [EC] Open blocked elevator door.
c) [EC] Wait for unblocking (doorway sensor signal).
d) [EC] Close elevator door.
e) (Continue with process step following door close.)

Audible alarm triggered

a) [Pa/Op] Press and hold in-car alarm button.
b) [EC] Sound alarm bell.
c) (Continue as usual, but with alarm on.)
d) [Pa/Op] Release alarm button.

e) [EC] Silence alarm bell.

Emergency stop triggered

a) [Pa/Op] Press in-car emergency stop button.
b) [EC] Close elevator doors.
c) [EC] Move elevator to default/recall floor.
d) [EC] Illuminate default/recall floor on all floor indicators.
e) [EC] Deluminate all buttons.
f) [EC] Open default/recall floor door and elevator car door.
g) [EC] Wait for shut-down (UC2) or mode change (UC3) to reset.

Weight overload detected (occurs only while doors are open)

a) [EC] Detect overload (interrupt from load sensor).
b) [EC] Sound audible indicator. (Do not close doors.)
c) [EC] Wait for weight reduction to limit (load sensor signal).
d) [Pa] Exit elevator car (until weight no longer over limit).
e) [EC] (Continue with close door step.)

4.3.5 UC5: Use elevator in AUTO mode – Drop-Off

Goal:	Passenger disembarks (dropped-off by) elevator car.
Pre-Cond.:	Elevator system is turned on and in AUTO mode, passenger is inside elevator car, doors are closed.
Post-Cond.:	Passenger exits on selected destination floor. Elevator car awaits next destination selection or pick-up request, doors are closed. Elevator car is located on selected destination floor.
Event:	Passenger (already in elevator car) selects destination floor.
System:	Elevator Controller (EC)
Actors:	Passenger (Pa)
Overview:	Description of EC behaviour in AUTO mode for passenger drop-off (and potentially pick-up once at the drop-off floor).
Related UCs:	UC4, UC3, UC2

Process Description:

1. [Pa] Select destination floor by pushing floor button on in-car button panel.
2. [EC] Move elevator car to destination floor.
3. [EC] Update in-car and corresponding entrance floor indicators at each floor.
4. [EC] Deluminate destination floor button.
5. [EC] Deluminate final floor up/down button if appropriate.
6. [EC] Open final floor door and elevator car door.
7. [Pa] Exit elevator car. (New passengers may board.)
8. [EC] Close current floor door and elevator car door after brief timeout.
9. [EC] Wait for signal from in-car button panel or up/down button panel at an elevator entrance.

Alternatives:

Passenger is actually Operator and switches elevator mode

2. [Pa→Op] Turn mode key-switch in elevator car.
3. [EC] Perform mode change as in UC3.

In-car door-open button pressed-and-held

9. [Pa] Press-and-hold door-open button.
10. [EC] Open current floor door and elevator car door
11. [Pa] Release door-open button.
12. [EC] Continue with default process step 9, above.

Exceptions:

System shut-down initiated mid-flight

a) [Op] Turn master key-switch in elevator control room to OFF position.
b) [EC] Perform shut-down sequence as specified in UC2.

Door blockage detected during attempt to close door

a) [EC] Detect door blockage (interrupt by doorway sensor).
b) [EC] Open blocked elevator door.
c) [EC] Wait for unblocking (doorway sensor signal).
d) [EC] Close elevator door.
e) (Continue with process step following door close.)

Audible alarm triggered

a) [Pa/Op] Press and hold in-car alarm button.
b) [EC] Sound alarm bell.
c) (Continue as usual, but with alarm on.)
d) [Pa/Op] Release alarm button.
e) [EC] Silence alarm bell.

Emergency stop triggered

a) [Pa/Op] Press in-car emergency stop button.
b) [EC] Close elevator doors.
c) [EC] Move elevator to default/recall floor.
d) [EC] Illuminate default/recall floor on all floor indicators.
e) [EC] Deluminate all buttons.
f) [EC] Open default/recall floor door and elevator car door.
g) [EC] Wait for shut-down (UC2) or mode change (UC3) to reset.

Weight overload detected (occurs only while doors are open)

a) [EC] Detect overload (interrupt from load sensor).
b) [EC] Sound audible indicator. (Do not close doors.)
c) [EC] Wait for weight reduction to limit (load sensor signal).
d) [Pa] Exit elevator car (until weight no longer over limit).
e) [EC] (Continue with close door step.)

4.3.6 UC6: Use elevator in HOLD mode – Drop-Off only

Goal:	Passenger embarks elevator car on *current* floor and disembarks on selected destination floor.
Pre-Cond.:	Elevator system is turned on and in HOLD mode (thus doors are open).
Post-Cond.:	Elevator car awaits next destination selection (pick-up requests are

ignored), doors are open. Elevator car is located on selected destination floor.

Event:	Passenger initiates trip with elevator in HOLD mode.
System:	Elevator Controller (EC)
Actors:	Passenger (Pa)
Overview:	Description of EC behaviour in HOLD mode.
Related UCs:	UC3, UC2

Process Description:

1. [Pa] Enter elevator car (recall that doors will already be open).
2. [Pa] Select destination floor by pushing floor button on in-car button panel. (Only first selection is accepted – cannot specify multiple destinations in HOLD mode.)
3. [EC] Close current floor door and elevator car door after brief timeout.
4. [EC] Move elevator car to destination floor.
5. [EC] Update in-car and corresponding entrance floor indicators at each floor.
6. [EC] Deluminate destination floor button.
7. [EC] Deluminate all up/down buttons.
8. [EC] Open final floor door and elevator car door.
9. [Pa] Exit elevator car. (New passengers may board.)
10. [EC] Wait for signal from in-car button panel.

Alternatives:

Passenger is actually Operator and switches elevator mode

2. [Pa→Op] Turn mode key-switch in elevator car.
3. [EC] Perform mode change as in UC3.

In-car door-open button pressed-and-held

3. [Pa] Press-and-hold door-open button.
4. [EC] Open current floor door and elevator car door
5. [Pa] Release door-open button.
6. [EC] Continue with default process step 3, above.

Exceptions:

System shut-down initiated mid-flight

a) [Op] Turn master key-switch in elevator control room to OFF position.
b) [EC] Perform shut-down sequence as specified in UC2.

Door blockage detected during attempt to close door

a) [EC] Detect door blockage (interrupt by doorway sensor).
b) [EC] Open blocked elevator door.
c) [EC] Wait for unblocking (doorway sensor signal).
d) [EC] Close elevator door.
e) (Continue with process step following door close.)

Audible alarm triggered

a) [Pa/Op] Press and hold in-car alarm button.
b) [EC] Sound alarm bell.
c) (Continue as usual, but with alarm on.)

d) [Pa/Op] Release alarm button.
e) [EC] Silence alarm bell.

Emergency stop triggered

a) [Pa/Op] Press in-car emergency stop button.
b) [EC] Close elevator doors.
c) [EC] Move elevator to default/recall floor.
d) [EC] Illuminate default/recall floor on all floor indicators.
e) [EC] Deluminate all buttons.
f) [EC] Open default/recall floor door and elevator car door.
g) [EC] Wait for shut-down (UC2) or mode change (UC3) to reset.

Weight overload detected (occurs only while doors are open)

a) [EC] Detect overload (interrupt from load sensor).
b) [EC] Sound audible indicator. (Do not close doors.)
c) [EC] Wait for weight reduction to limit (load sensor signal).
d) [Pa] Exit elevator car (until weight no longer over limit).
e) [EC] (Continue with close door step.)

4.3.7 UC7: Use elevator in SERVICE mode – Suspended

Note: This use case is redundant and can be eliminated (as a standalone use case). However, it is included simply for completeness and *explicit* description of elevator behaviour while in SERVICE (recall) mode.

Goal:	EC holds elevator car (for servicing).
Pre-Cond.:	Elevator system is turned on and in SERVICE mode.
Post-Cond.:	Elevator system is turned on and in SERVICE mode.
Event:	Elevator enters/remains in SERVICE mode.
System:	Elevator Controller (EC)
Actors:	Elevator Controller (EC)
Overview:	Description of EC behaviour in SERVICE mode.
Related UCs:	UC3, UC2

Process Description:

1. [EC] Wait for in-car mode key-switch position change (i.e. do nothing until elevator mode is switched out of SERVICE).

Exceptions:

System shut-down initiated mid-flight

a) [Op] Turn master key-switch in elevator control room to OFF position.
b) [EC] Perform shut-down sequence as specified in UC2.

Audible alarm triggered

a) [Pa/Op] Press and hold in-car alarm button.
b) [EC] Sound alarm bell.
c) (Continue as usual, but with alarm on.)
d) [Pa/Op] Release alarm button.
e) [EC] Silence alarm bell.

4.3.8 Use Case Diagram

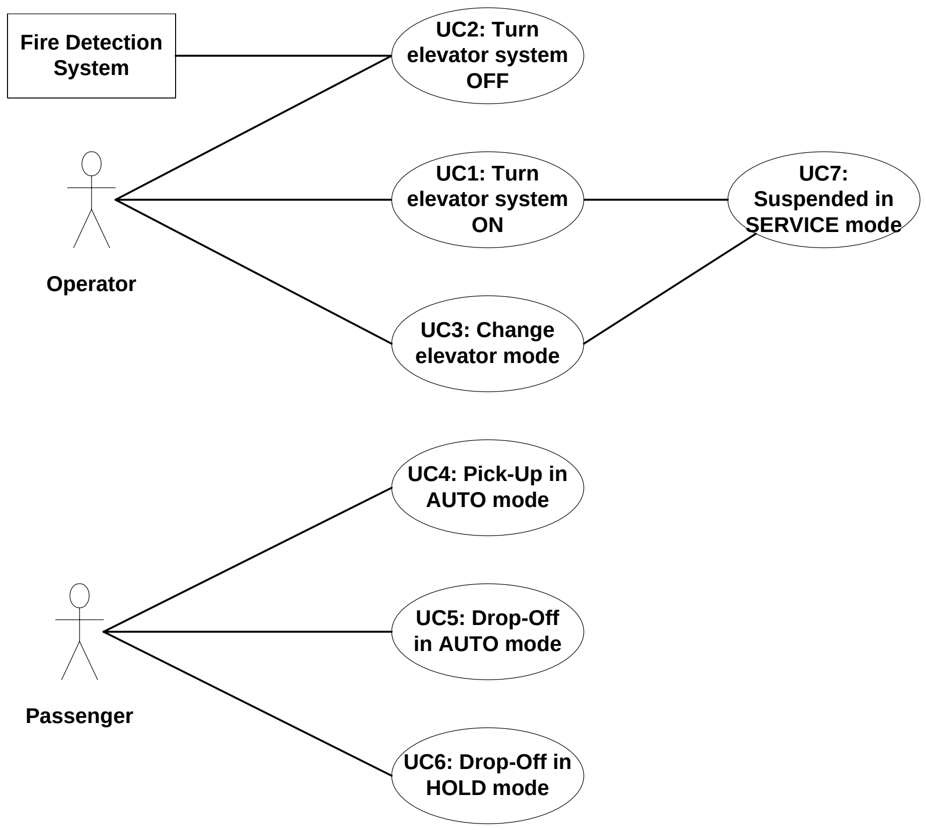
Fire Detection System
UC2: Turn elevator system OFF
UC1: Turn elevator system ON
UC7: Suspended in SERVICE mode
Operator
UC3: Change elevator mode
UC4: Pick-Up in AUTO mode
UC5: Drop-Off in AUTO mode
Passenger
UC6: Drop-Off in HOLD mode

Figure 37: Elevator Controller Use Case Diagram