LICENSE PLATE RECOGNITION SYSTEM

Group Members

Name:

Muhammad Waseem

Roll No:

BSCS-F16-E005

Name:

Haseeb-ur-Rehman

Roll No:

BSCS-F16-E006

INTRODUCTION

Automatic Number Plate Recognition (ANPR) system is an important technique used in intelligent transport System (ITS), which provides services to different modes of Transport and traffic management. ANPR system is an advanced machine vision technology used to identify vehicles by their number plates without direct human intervention. It is a challenging problem in the field of machine vision and automation with various applications such as parking lot ticketing system, automated hands free toll collection, automated vehicle access in secure establishments, law enforcement, border security and custom security etc. The whole ANPR system is generally framed into 3 steps:

1. Number Plate Detection,
2. Character Segmentation and
3. Character Recognition.

In the Number plate detection work, many techniques have been proposed such as: statistical analysis of Discrete Fourier Transform (DFT) of the plate signal , global edge features and hear like features , wavelet transform and EMD (Empirical Mode Decomposition) analysis , region and edge based methods and sliding window technique for efficient number plate localization based on discrete wavelet transform Morphological In the characters segmentation module, there are also some techniques to address this work such as: edge detection, color model transform, connected components analysis an intelligent framework that outlines character of car number plate by various illuminations

Horizontal projection multi-clustering algorithm threshold and connected components horizontal and vertical projections Color reverse, vertical edge detection, horizontal projection histogram, vertical projection morphology operation and connected components In the characters recognition module, there are also many techniques to address this work such as: color image processing hidden Markova model support vector machine [9], multi-cluster and multilayer neural networks Least squares support vector machines multi-layer perceptron network template matching fuzzy multilayer neural Bayesian framework radial basis function neural networks. Although, there are many method proposed for ANPR system.

But, there is not a single method can provide satisfactory performance in all the applications in various complicated background such as: uncertainty of edges, various types of plate, the plate is small, dim lighting low or high illuminated images, types of plate, colors, character fonts, syntax, size, angle of the number plate, weather and environment, multi-rows, Kannada number plates.

To cope with these limitations, we have considered the Indian NP to propose a new method for the Indian ANPR system, which satisfied for all types of Indian NP. This paper is organized as follows: section II introduces Number plate Detection, section III shows the characters segmentation and recognition, section IV shows experimental results and section V is conclusions and the last is references.

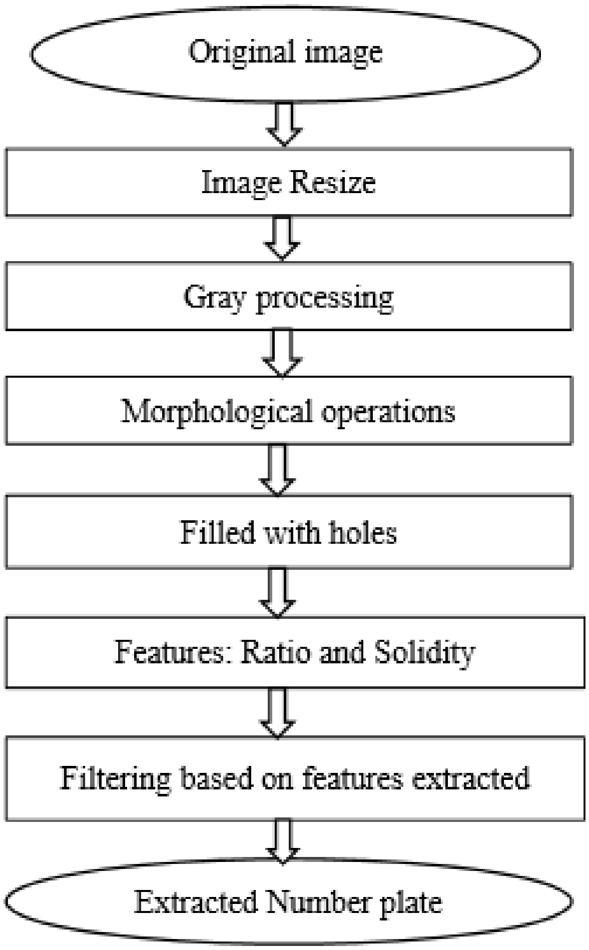
Table of Contents

NUMBER PLATE DETECTION

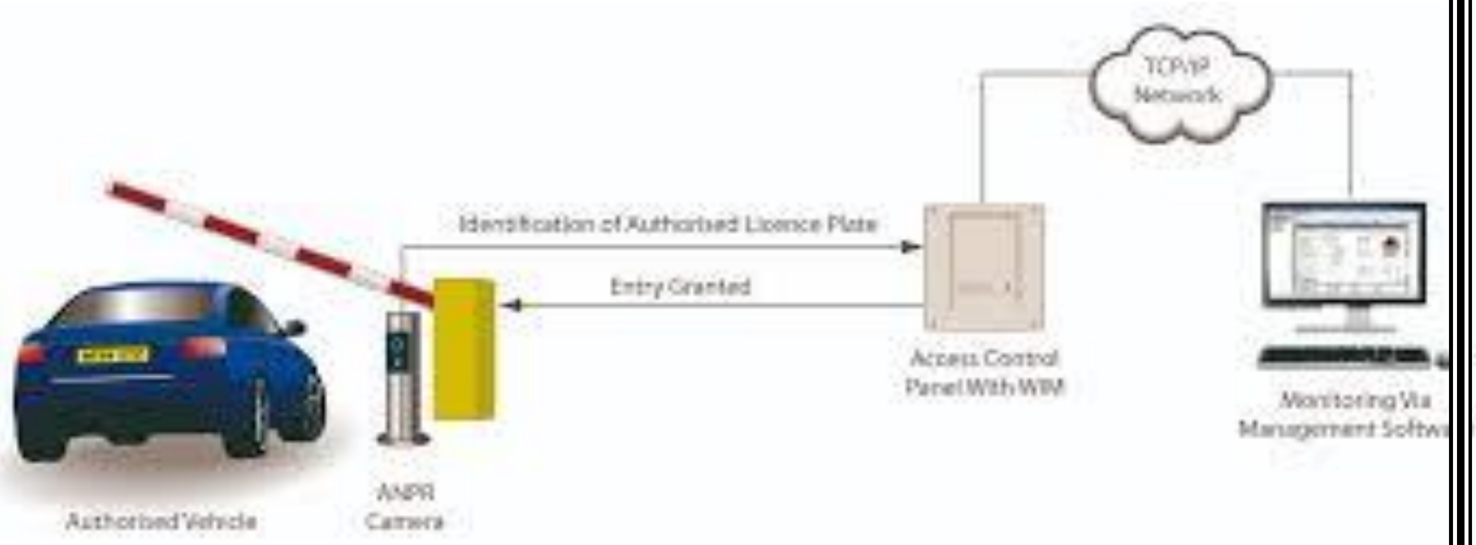
The Plate detection phase aims at identifying the number plate area in the image. It is often composed of image pre-processing or plate enhancement phase that helps to enhance the signal in the number plate area and attenuate it elsewhere

Many difficulties occur during the detection and extraction of number plate due to the following reasons:

1. The efficiency of extraction is affected by complex background in an input image.
2. Variations in the position of number plate in different vehicles
3. Different size plates due to camera distance and zoom factor
4. Plates may have various characters and background colors due to different types of plates
5. Unwanted characters, frames and screws introduce confusion
6. Different font size, font style and different languages
7. Occlusion- plates may be obscured by dirt
8. Inclination – plates may be tilted Here we have considered all the above difficulties and worked on all of them. Our algorithm is able to recognize the number plate efficiently in all such difficulties as listed above.



قطر ١٦٥٥١١


TCP/IP Network
Identification of Authorised Licence Plate
Entry Granted
Access Control Panel With WIM
Monitoring Via Management Softw
Authorised Vehicle
ANPR Camera

Person Statement of Requirements

Statement of Goals

The main goals of this software development project is to maximize the occupancy and revenue of a parking area and develop a user-friendly mechanism and automatically extract number from vehicle number plate.

Problem Statement

The problems that are faced is as follows:

In this project, the main problem is to design and develop an automatic number plate recognition system for the application in a toll payment and for security purpose. With this project, the vehicles number plate will be automatically captured and the system will do all the images preprocessing method and the output will display using GUI.

Another side problem is to upgrade, improve and rewrite the algorithms to make sure it is convenient with present technology. It is challenge to the system to extract the images because the normal standard license plate number already has its own standard size and pattern.

Proposed Solution

In order to increase profits and reduce personnel costs a computerized system will be put into place to address all three problems suggested above.

The first problem of signaling to persons who may drive by the parking area and not know if any parking is available can be solved by implementing a display at the entrance of the parking area indicating if room is available to house persons driving in off the street. The second problem, congestion inside of the parking area, can be solved by assigning persons parking space numbers of spots in the parking area, so that the person will always know where the next available spot is. Finally, the third problem of not knowing if parking will be available for a given date and time will be solved by implementing a web-based system (a website) that will allow persons to create accounts and create reservations using that account. These reservations will be simple guarantees that parking will be available for the selected date and time, removing the fear of being stuck in an area with no place to park.

To implement these changes, several new pieces of hardware will need to be adapted for use in the parking area. The system is based upon a multi-level parking area that has a vehicle elevator between levels. Cars may only enter the parking area through the elevator, and can leave through a one-way exit ramp. This elevator can only accommodate passenger vehicles, so large trucks and other high-capacity vehicles will need to be excluded from parking here.

The remaining hardware will handle detecting if a vehicle has entered the parking area, detecting a vehicle leaving the parking area, and determining if a vehicle is occupying a spot in the parking area.

The final piece of our solution is a delineation between person types: registered persons versus walk-ins. Registered persons have created accounts with the parking parking area's online service, and may make reservations in advance. Walk-ins are not associated with the online service and can only park when they drive in off the street. Since this business will derive a majority of its profits from repeat business of registered persons, we elect to have the ground level of the parking parking area allow only walk-ins to park, whereas the upper levels will be reserved for registered persons fulfilling reservations. It is possible that at some point in the future these restrictions could be eased, however the initial software solution will be kept simple.

The parking and reservation system will be nicknamed “Park-A-Lot”.

Devices

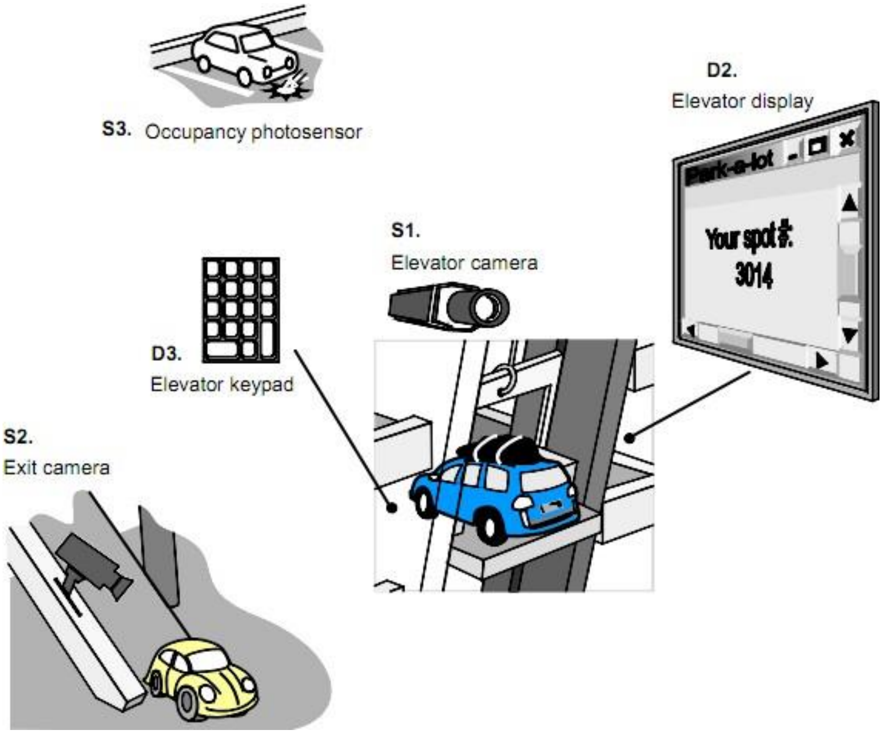
To accomplish the goals state above, Park-A-Lot will implement the following devices:

S1 & S2. The cameras will be installed to act as license-plate readers. S1 is the camera at the elevator and S2 is the exit camera. The cameras will be using the license-plate recognition system that are often used in tolls. The basic idea is that when a car arrives at the elevator platform, S1 reads the registration number and later on S2 will read the registration number of the vehicle leaving.

S3. There will be a sensor installed at the gate. This will help us determine whether the spot is available or not.

D1. The digital display at the elevator will display different messages according to the specific situation. The messages that might appear in this display are: denied access for non-registered persons or change/edit in the reservation of a registered person.

Although, this is not specifically a device, there will be a barrier at the gate so that no vehicles will try to enter.



Assumptions

Although our software solution will attempt to cover as many situations and scenarios as possible, the following general assumptions will be made.

A1. The camera's license-plate recognition system does not fail, meaning it is correct all the time, regardless if the plate is dirty or has damages. Also, if the registration number is not recognized by the system then is is assumed that the car does not correspond to any registered person.

A2. If the elevator camera's license-plate recognition system does not identify the registration number and the person fails to provide a correct one then the system will display a message on elevator display telling them to back up from the elevator. If this occurs then it is assumed that the person obediently leaves the elevator.

A3. The sensors that detect the occupancy of the spots work correctly all the time, disregarding any malfunctioning of the devices. Also, every time a sensor detects occupancy it is because a vehicle is there and not another object.

A4. The elevator will lift the car to the corresponding deck and will not make any mistakes.

A5. The person will not fail to park at his or her assigned parking spot.

A6. If the system recognizes the vehicle's registration number then it is assumed that the person driving the car is a registered perso

Image acquisition:

The first step is to acquire the input image of the vehicle. Images are captured by a digital camera at various distances from the camera. The better the quality of the input images are, the better conditions the number plate recognition algorithm has, and thus the higher number plate recognition accuracy can be expected to be achieved.

Image Resize:

The image to be segmented is rescaled. Resizing images involves reducing the pixel dimensions and using higher compression. Both of these actions reduce file sizes dramatically.

Gray processing:

Grayscale Image is known as an intensity, gray scale, or gray level image. Array of class unit8, unit16, single or double whose pixel values specify intensity values. For single or double arrays, values range from [0, 1]. For unit8, values range from [0, 255]. For unit16, values range from [0, 65535]. For int16, values range from [32768, 32767]. Gray processing is very important step in image processing, its result is the foundation of later steps. Resized image is converted into gray scale image.

Binary images might be too simple and cannot represent the picture character. Color images are too complex and affect the processing speed and they do not help in identifying important edges or other features. The basic concept of gray conversion is to eliminate hue and saturation image while maintaining its luminance. The true color to gray-scale conversion is performed by Gray=0.299r+0.587g+0.114b---------- (1) Where Gray is the new pixel value and r, g, and b are red, green and blue values of the original pixel respectively.

Morphological operation:

Morphological image processing is a collection of non-linear operations related to shape or morphology of features in an image. Mathematical Morphology is a way of nonlinear filters, which could be used for image processing as well as noise suppression, feature extraction, edge detection, image segmentation, shape recognition, texture analysis, image restoration and reconstruction, image compression etc.

Dilation:

It is defined as the maximum value in the window. Hence the image after dilation will be brighter or increased in intensity. It also expands the image and mainly used to fill the spaces. It is the operation of lengthening or thickening in binary image.

Erosion:

It is just opposite to dilation. It is defined as the minimum value in the window. The image after erosion will be darker than the original image. It shrinks or thins the image.

Opening and Closing:

Both parameters are formed by using dilation and erosion. In opening, firstly image will be eroded and then it will be followed by dilation. In closing, the first step will be dilated and then the result of this is followed by erosion. Opening operation generally makes the contour of objects smoother, and disconnects narrow, discontinuous and remove thin protrusions. Closing operation makes an outline smooth, it usually eliminate discontinuity and narrows long, thin gap, clears up small holes and fill the ruptures of the contour line. Here morphologically open binary image is applied for the binary image from previous stage. It removes small objects. It removes from a binary image all connected components (objects) that have fewer than P pixels, producing another binary image BW2. This operation is known as an area opening. The default connectivity is 8 for two dimensions, 26 for three dimensions. To remove unwanted objects or regions, small dots and very big areas from an image, this operation is used here. We could have used close operation to perform removal of small and big objects from an image but comparatively this method gives better results.

Filled holes:

Hole filling may be defined as a background region surrounded by a connected border of foreground pixels. This process makes the detected regions from the previous step a solid object by removing black pixels on white object. This helps in detecting the number plate region.

Feature extraction

Feature extraction in image processing is a technique of redefining a large set of data into a set of features of reduced dimensions. The result of the previous stage is the retention of a few candidate regions as possible number plates. Now we have to find which candidate is the true region and so two important features are used to discard the wrong candidate regions. These features are aspect ratio, and solidity.

1. **Area:**

The area is determined by counting the total number of non-zero pixels within the image. Area: A=H x W

2. **Aspect Ratio:**

Aspect ratio describes the relationship between the width and height of an image. Aspect Ratio = H/W Where His Height and W is Width

3. **Solidity:**

The ratio of actual area and convex hull area is known as solidity and is an essential for true number plate detection. S=Area/Convex Area The convex area is the area inside the convex hull of the 2D object. Convex area is the product of Height and Width of the segmented region. The Aspect Ratio and

Solidity are calculated for each segmented regions or objects detected in the previous stage. These values help in detecting the true number plate region.

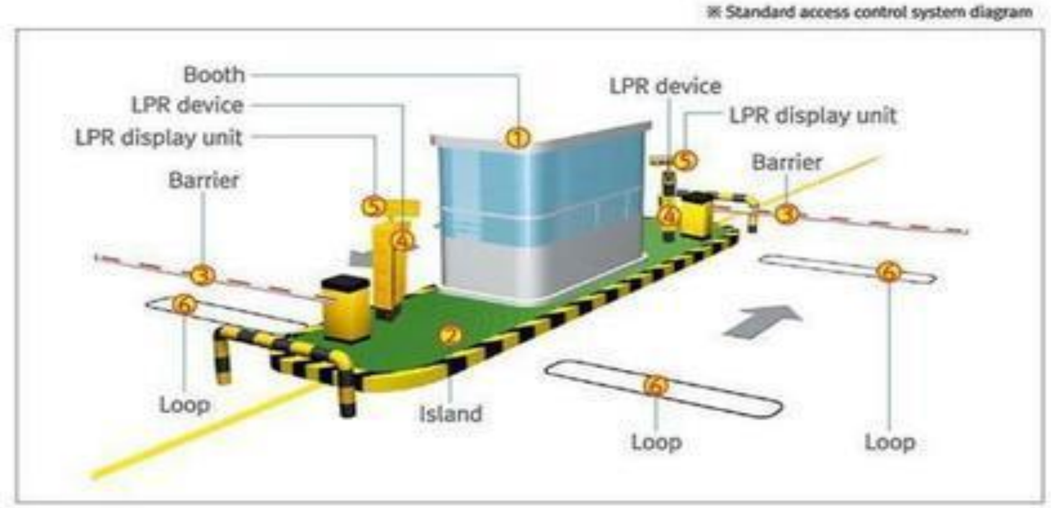
Filtering:

Based on these two feature values true number plate region is detected. The aspect ratio and solidity values are multiplied pixel by pixel level (AND operation). Thus true ROI is detected.

Extract Number Plate using bounding box technique: The bounding box technique is used to extract the number plate region. The minimum row, maximum row, minimum column and maximum column components are used to locate the number plate in terms of bounding box drawn over the binary image. Here in this step the number plate region is separated out from the whole vehicle image using these vertices or coordinate values.

User Interaction Requirements

Below is an overview diagram of the Park-a-Lot system concept.



The system will have the following requirements and design specifications.

1. A database that will contain:
    a. All registered persons' information;
    b. All the parking reservations (past, current and future);
    c. All vehicles registered to person accounts;
    d. A record of all person transactions (i.e. parking area usage history, past reservations, punctuality or missed reservations);

    A system administrator role which will be allowed to

    e. View the registered persons' profiles and person statistics

Pre-processing:

The detected number plate may contain noise, skew and there may be broken and degraded characters. So, the image is pre-processed to remove the noise, line skew, and to correct the broken and degraded characters. Firstly, the extracted number plate is converted into binary image for further processing. Then, morphological operations like filled holes are used with removal of noise to get a solid object. The noisy, broken characters can be easily detected and could be correctly recognized if the characters are in white pixels with black background. Therefore the black and white number plate image is inverted to get white characters on black background. This is called as complimented image or inverted image

Thresholding:

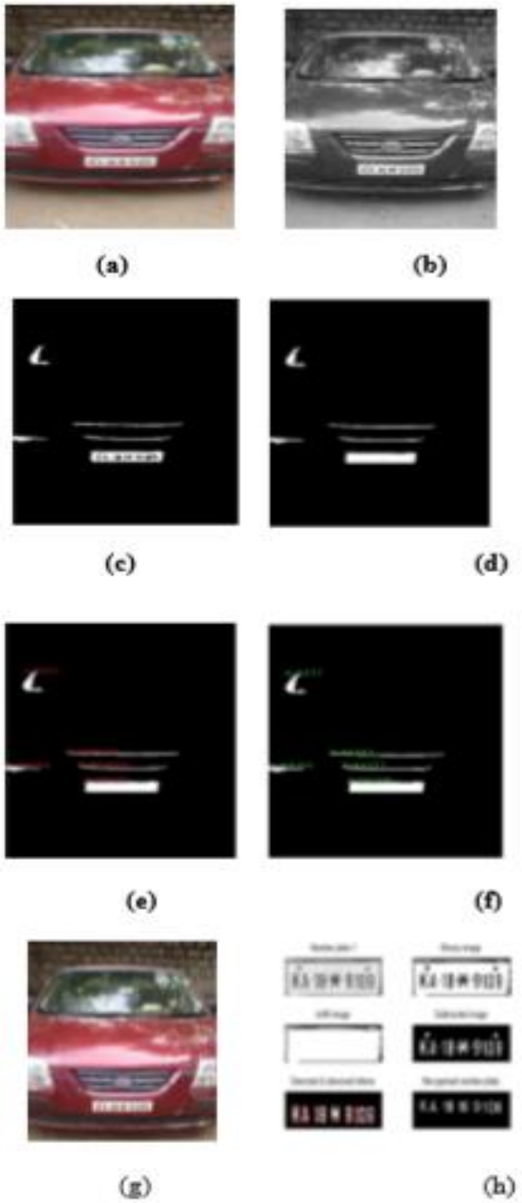
Thresholding is non-linear operation that converts a gray scale image into binary image where
the two levels are assigned to pixels that
are below or above the specified threshold value. The threshold of an image with correct gray scale value is calculated for the purpose of separating the object of interest from background. Thresholding is important to provide sufficient contrast of an image such that, different level of intensity between foreground and background are taken into consideration. The purpose of thresholding is to extract those pixels from some image which represent an object (either text or other line image data such as graphs, maps). For computational purposes gray scale improves the quality of an image and the later computational processes

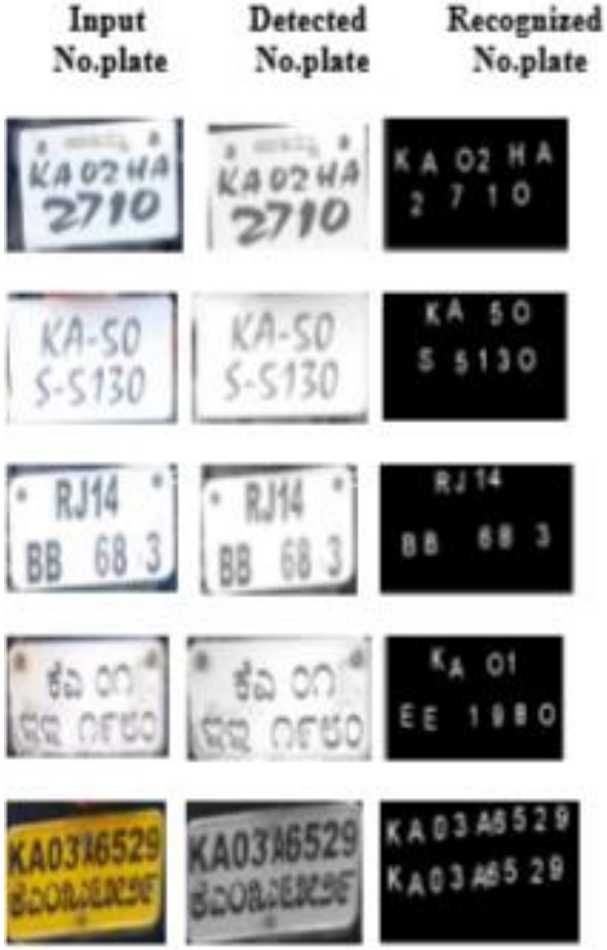
Character bounding box for Segmentation:

Bounding boxes for connected components are the properties of the labeled connected component regions. A bounding box of a labeled region is a rectangle that just encloses the region completely. When a specific bounding box is determined for a connected region, the coordinates of the corners of the bounding box and its width and height are available. A bounding box completely specifies the boundaries of the corresponding connected component. Character Recognition using ANN: MLP Network: A Multi-layer perception is a feed-forward artificial neural network model that maps sets of input data onto a set of appropriate outputs. A MLP consists of multiple layers of nodes in a directed graph, with each layer fully connected to the next one. These networks have been applied to distinct areas, performing tasks such as function fitting and pattern recognition problems, by using the supervised training with an algorithm known as error back propagation. In this paper, we proposed an improved method based on MLP neural network and backpropagation algorithm for training to recognize characters & numbers in the Indian NP.

RBF Network:

Radial Basis Function Neural Network (RBF) is a local approximation neural network and it is better than Back Propagation (BP) Neural Network in such abilities as approximation


(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)


Input No.plate
Detected No.plate
Recognized No.plate
KA 02 HA 2710
K A 02 H A 2 7 1 0
KA-50 S-5130
KA 5 0 S 5 1 3 0
RJ14 BB 68 3
R J 14 B B 6 8 3
KA 01 EE 1 9 8 0
KA03A6529
K A 0 3 A6 5 2 9
K A 0 3 A6 5 29

Glossary of Terms

Person ID - the email address a person uses to register with the online Park-A-Lot system.

Elevator Camera - an image capturing camera capable of utilizing image recognition to determine the license plate number on a vehicle inside the parking parking area elevator.

Elevator Display - an LCD screen located inside the parking parking area elevator for displaying information.

Exit Camera - same as elevator camera, and capable of using image recognition on license plates of vehicles exiting the parking area.

No Vacancy - The parking area is full and is not walk-in parking at this time.

Rain Check Credit - it is a credit given to the person when he or she arrives at the parking parking area and no spots are available to park in.

Reservation - an arrangement to park in a parking parking area for a fixed amount of time at a certain fee per hour.

Recurring Reservation - a reservation, made in advance with a variable grace period, that occurs on some regularly repeating schedule.

Reservation Parking - when a person parks at the parking area to fulfill a pre-existing reservation (either recurring or confirmed).

Spot Sensors - sonar sensors capable of determining whether a parking spot inside the parking area is occupied with a car or not.

System Administrator - person who will have deep access to the system, and be able to alter business requirements such a parking prices.

Functional Requirements Specification

Stakeholders

A stakeholder is anyone who has interest in this system (users, managers, sponsors, etc.).

1. Registered Person
2. Unregistered Person
3. System Administrator
4. Security Personnel

Actors and Goals

An actor is anyone who will directly interact with the system. The two types of actors are initiating and participating.

1. Initiating
    a. Registered Person
    b. Unregistered Person
    c. System Administrator
    d. Timer
2. Participating
    a. Elevator Keypad
    b. Elevator Display
    c. Elevator Camera
    d. Spot Sensors
    e. Exit Camera
    f. Database
    g. Event Log

Use Cases

Casual Description of Use Cases

If the person is an unregistered the system will check for open spots on the ground floor. The person will then swipe his/her credit card before driving though to park.

Finally, if the person arrives early to his or her reservation, the following applies:

1. Check their license plate for a future reservation, and if found ask if they are there for that reservation.
    a. If yes, and if the upcoming reservation is within some time limit (30 minutes before the start time), then if there is available parking they may park, and be charged for the additional time at the rate per hour of their soon to be occurring reservation.
    b. If the user arrives very early (more than 30 minutes before start time), they will have to park as a walk-in until the start time of their reservation. This would also require us to make a note that the walk-in reservation would coincide directly with their upcoming scheduled reservation, meaning that the person should not be expected to leave the walk-in parking and re-enter the parking area to fulfill their scheduled reservation. We will assume the walk-in parking will continue directly to the actual reservation scheduled. The billing should be at a walk-in rate for the walk-in, and at the regular rate for the reservation.
    c. Finally, if no spots are available, the person will be told they cannot park at this time.
2. If the license plate isn't recognized, we prompt for credentials, and follow the instructions given in (1) if the person is registered with a reservation in the future for that day.

UC-1 Manage Account

A registered person wants to change their account details (email, password, address, credit card info, etc.).

UC-2 Manage Parking area

A system administrator wants to manage the parking area remotely. After being authenticated by the system, the administrator will be presented with options to set parking prices, inspect usage history, as well as view current usage. All options are sub-use cases described later.

UC-3 Register Vehicle

A registered person wants to register a vehicle for his/her account. After first being authenticated by the system, the system will show a form, which the user will fill out and submit. The system will then validate the submitted information (license plate number, state, color) and store it in the database, assigning the user a unique vehicle id.

UC-4 Authenticate User

A registered person wants to log in to the system. The system will present a log in form (email, password) which the person will fill out and submit. The system will search for and find the person in the database. The system will then start a session for the user, which will last until the user logs out or closes his/her brows

Fully-Dressed Description of Use Cases

Use Case UC-1:	**Reserve**
Related Requirements:	REQ15, REQ16, REQ17
Initiating Actor:	Registered Person
Actor's Goal:	To reserve a parking spot for a future date and time.
Participating Actors:	Database
Preconditions:	Registered person is currently logged into the System. *include::AuthenticateUser* (UC-10)
Postconditions:	The System reserves the requested date and time for the person in the Database.

Flow of Events for Main Success Scenario:

→ 1. **Registered Person** selects menu option "Make Reservation".

→ 2. **Registered Person** selects the desired date, start time, and end time for the reservation, along with any additional options (i.e. for a recurring reservation the person may wish to extend the grace period).

← 3. **System** (a) checks the reservation **Database** for available reservations, (b) notifies **Registered Person** that the reservation is made, and (c) updates the **Database** to include the new reservation.

Flow of Events for Extensions (Alternate Scenarios):

3a. **System** cannot find an available reservation for the specified date and time.

← 1. **System** (a) logs the attempted reservation and (b) signals to the **Registered** person

→ 2. **Registered** person selects a different date and time to make a reservation.

3. Same as in step 4.

3b. **System** finds a conflicting reservation from **Registered Person**'s account.

← 1. **System** notifies person that he or she already has a reservation or part of a reservation during that date and time.

→ 2(a). **Registered** person can choose to cancel the existing reservation in favor of the new reservation. *include::EditReservation* (UC-7). Then same as in step 4.

→ 2(b). **Registered** person can choose to book overlapping reservations. Then same as in step 4.

Use Case UC-2: **Park**

Related Requirements:	REQ1, REQ2, REQ3, REQ4, REQ5, REQ6, REQ7, REQ19, REQ20
Initiating Actor:	Registered person
Actor's Goal:	To park in the parking area.
Participating Actors:	Elevator Display, Elevator Camera, Elevator Keypad, Spot Sensor Exit Camera, Database
Preconditions:	The elevator is currently empty.
Postconditions:	System marks reservation as completed in database.

Flow of Events for Main Success Scenario:

→ 1. person enters the elevator.

2. **Elevator Camera** recognizes license plate of vehicle.

← 3. **Elevator Display** displays any reservations returned from the **Database** the person may have for the current date.

→ 4. person selects a reservation.

← 5. **System** (a) assigns an optimal spot to the **Person** and (b) displays the parking spot on the **Elevator Display**.

→ 6. **Person** exits the elevator.

← 7. **Spot Sensor** notifies the **System** when a vehicle is parked in the assigned parking space.

← 8. **Spot Sensor** notifies the **System** when a vehicle is no longer parked in the assigned space.

9. **Person** exits parking parking area and is recorded by **Exit Camera**.

Flow of Events for Extensions (Alternate Scenarios):

2a. **Elevator Camera** fails to recognize license plate of vehicle.

← 1. **Elevator Display** prompts **Person** for either (1) a registered person ID or (2) a method of payment for a walk in.

→ 1(a). **Registered Person** enters ID number, password, and license plate number into **Elevator Keypad**.

1(b). *include::AuthenticateUser* (UC-10)

1(c). Same as in step 3.

→ 2(a). **Person** slashes credit card to pay.

← 2(b). **Elevator Display** prompts user for reservation length.

→ 2(c). **Person** enters reservation duration into **Elevator Keypad**. 2(d).

Same as in step 5.

3a. **Person** arrives early for reservation.

1. If the reservation start time is within 30 minutes of the current time, then

→ 1(a). If parking spaces are available in the parking area, then modify the start time of the reservation by 30 minutes so that **Person** may park immediately.

1(b). Same as in 3.

2.If the reservation start time is more than 30 minutes from the current time, then

2(a). Same as in (3b).

2(b). Make note that the **Person** will not be required to leave the assigned spot to fulfill his or her upcoming reservation.

3b. **Person** does not have any existing reservations.

← 1. **System** prompts user for reservation length.

→ 2. **Person** enters reservation duration into **Elevator Keypad**.

3. Same as in step 5.

3c. **Person** arrives after grace period has expired.

← 1. **Person** is informed that grace period has expired and is offered the chance to park as a walk-in if space is available.

2. Same as in 5.

Use Case UC-3: **Manage Account**

Related Requirements:	None
Initiating Actor:	Registered Person
Actor's Goal:	To edit the details of a person's account.
Participating Actors:	Database
Preconditions:	Registered Person is currently logged in to the system. *include::AuthenticateUser* (UC-10)
Postconditions:	Changes to Database are committed.

Flow of Events for Main Success Scenario:

→ 1. **Registered Person** selects menu option "Manage Account".

← 2. **System** (a) displays current user account details, and (b) prompts **Registered Person** to make changes to desired fields.

→ 3. **Registered Person** makes the necessary changes to the form.

← 4. **System** verifies that the changes made are valid.

Use Case UC-4: **View Reservations**

Related Requirements:	REQ10
Initiating Actor:	Registered Person
Actor's Goal:	To view existing reservations and edit any reservations.
Participating Actors:	Database
Preconditions:	Registered Person is currently logged in to the system. *include::AuthenticateUser* (UC-10)
Postconditions:	Changes to Database are committed.

Flow of Events for Main Success Scenario:

→ 1. **Registered Person** selects menu option "View Reservations".

← 2. **System** displays any active or future reservations returned from the **Database** the person may have and displays options for editing reservations.

→ 3. **Registered Person** selects one of the reservations or multiple reservations (if they are recurring) to edit and follows the corresponding instructions or does not select any reservation.

← 4. **System** (a) stores the changes made in the Database, and (b) signals to the **Registered Person** the successful change.

Flow of Events for Extensions (Alternate Scenarios):

3a. Selected activity entails editing the reservation.

1. *include::EditReservation*(UC-7).

Use Case UC-5: Register

Related Requirements:	REQ0
Initiating Actor:	Unregistered Person
Actor's Goal:	To create an account and become a Registered Person.
Participating Actors:	Database
Preconditions:	Unregistered Person has a valid email address with which to register.
Postconditions:	The System stores all of the newly Registered Person's information in the Database.

Flow of Events for Main Success Scenario:

→ 1. **Unregistered Person** accesses **System** and selects menu option "Create Account".

→ 2. **Unregistered Person** fills in personal info: name, address, state, zip, phone number, email address, password, billing credit card number and submits the info.

← 3. **System** (a) checks that all fields have been filled in, (b) verifies the email address is valid and unique within the **Database**, (c) verifies the credit card information is valid, and (d) updates the database to include the new **Registered Person**.

→ 4. New **Registered Person** can now make reservations in advance.

Flow of Events for Extensions (Alternate Scenarios):

3a. **System** identifies that not all of the fields have been filled in on the registration form.

← 1. **System** (a) detects error and (b) signals to the **Unregistered Person** that they must complete the form and resubmit.

→ 2. **Unregistered Person** fills in the missing data fields and resubmits the form.

3. Same as in step 3.

3b. **System** identifies that email address is invalid or has already been registered with the website

← 1. **System** (1) notifies the **Unregistered Person** that the email address is invalid prompts the **Unregistered Person** to change that information or (2) detects that the email address has already been registered in the system and alerts the **Unregistered Person** that the email address has already been registered and to attempt to log into the account.

→ 1(a). **Unregistered Person** changes the email address field and resubmits the form.

1(b). Same as in step 3.

2(a). **Registered Person** leaves registration area.

3c. **System** could not verify the credit card information.

← 1. **System** prompts the **Unregistered Person** to re-enter their credit card information.

→ 2. **Unregistered Person** changes the information in the credit card field and resubmits.

3. Same as in step 3.

Use Case UC-6: **Manage**

Related Requirements:	REQ13
Initiating Actor:	System Admin
Actor's Goal:	To set parking prices or inspect usage history.
Participating Actors:	Database, Event Log
Preconditions:	System Admin is currently logged in to the system. *include::AuthenticateUser* (UC-10)
Postconditions:	Changes to System are committed and System Admin is logged out.

Flow of Events for Main Success Scenario:

→ 1. **System Admin** selects menu option “Manage Parking area”.

← 2. **System** displays options for: (a) setting prices, (b) inspecting parking usage history.

→ 3. **System Admin** selects one of the options from Step 2 and performs management activities.

→ 4. **System Admin** commits changes to **System**.

5. **System** verifies and commits changes to **Database**.

6. **System Admin** logs out of his or her account.

Flow of Events for Extensions (Alternate Scenarios):

3a. Selected activity entails setting parking parking area prices.

1. *include::SetPrices* (UC-11).

3b. Selected activity entails viewing access history.

Use Case UC-7: **EditReservation** (sub-use case)

Related Requirements:	REQ8, REQ9
Initiating Actor:	Registered Person
Actor's Goal:	To extend an existing reservation.
Participating Actors:	Database
Preconditions:	Some reservations exist for the Registered Person in the Database.
Postconditions:	Extended reservation is marked as extended in the Database and the start and/or end time is updated accordingly.

Flow of Events for Main Success Scenario:

← 1. **System** (a) displays reservation(s) that the **Registered Person** selected, and (b) prompts the **Registered Person** to change the end time or cancel the reservation(s).

→ 2. **Registered Person** makes the desired selections / changes.

← 3. **System** (a) checks the database to see if the extensions can be made, and (b) notifies the **Registered Person** that the reservations have been extended.

4. **System** updates the database to include the changes to the reservation(s).

Flow of Events for Extensions (Alternate Scenario):

3a. **System** identifies that the extension cannot be made due to reservation conflict.

← 1. **System** notifies the **Registered Person** that the parking deck is completely booked and the reservation cannot be extended.

2. Same as in step 1 above.

Use Case UC-8: **RegisterVehicle**

Related Requirements:	REQ14
Initiating Actor:	Registered Person
Actor's Goal:	To register a vehicle.
Participating Actors:	Database

Preconditions:	Registered Person is currently logged in to the system. *include::AuthenticateUser* (UC-10
Postconditions:	The system stores the new vehicle information in the database.

Flow of Events for Main Success Scenario:

→ 1. **Registered Person** selects menu option "Register Vehicle".

→ 2. **Registered Person** enters the license plate number, the state, and the color of the vehicle.

← 3. **System** verifies that the license plate number is valid based on the state selected.

4. **System** stores the new vehicle information in the database.

Flow of Events for Extensions (Alternate Scenario):

3a. **System** identifies that the license plate number is not valid.

← 1. **System** notifies the **Registered Person** that the license plate number is not valid.

2. Same as in step 2 above.

Use Case UC-9:	**EditVehicle**
Related Requirements:	REQ14
Initiating Actor:	Registered Person
Actor's Goal:	To register a vehicle.
Participating Actors:	Database
Preconditions:	Registered Person is currently logged in to the system. *include::AuthenticateUser* (UC-10
Postconditions:	The system stores the new vehicle information in the database.

Flow of Events for Main Success Scenario:

→ 1. **Registered Person** selects menu option "Register Vehicle".

→ 2. **Registered Person** enters the license plate number, the state, and the color of the vehicle.

← 3. **System** verifies that the license plate number is valid based on the state selected.

4. **System** stores the new vehicle information in the database.

Flow of Events for Extensions (Alternate Scenario):

3a. **System** identifies that the license plate number is not valid.

← 1. **System** notifies the **Registered Person** that the license plate number is not valid.

2. Same as in step 2 above.

Use Case UC-10: **AuthenticateUser** (sub-use case)

Related Requirements:	None
Initiating Actor:	Registered Person, System Admin (collectively User)
Actor's Goal:	To be positiviely identified by the system.
Participating Actors:	Database
Preconditions:	The database contains user account information.
Postconditions:	None

Flow of Events for Main Success Scenario:

← 1. **System** prompts the user for their person ID and password.

→ 2. **User** supplies a valid person ID and password.

← 3. **System** (a) verifies that the person ID and password are valid, and (b) signals to the user the identification validity.

Flow of Events for Extensions (Alternate Scenarios):

2a. **User** enters an invalid person ID and password combination.

← 1. **System** (a) detects error, (b) marks a failed attempt, and (c) and signals to the **User** the credentials are invalid.

← 1(a). **System** (a) detects that the count of failed attempts exceed the maximum number, (b) informs the user to try again in 15 minutes, and (c) exits the screen.

→ 2. **User** supplies a valid person ID and password.

3. Same as in step 3.

Use Case UC-11:	**Set Prices** (sub-use case)

Related Requirements:	REQ13
Initiating Actor:	System Admin
Actor’s Goal:	To set the parking prices and penalty fees for a particular parking area.
Participating Actors:	Database
Preconditions:	System Admin is currently logged into the system. *include::AuthenticateUser* (UC-9)
Postconditions:	Price changes are stored in the Database.

Flow of Events for Main Success Scenario:

→ 1. **System Admin** selects the menu option “Set Prices”.

← 2. **System** prompts for the parking area location.

→ 3. **System Admin** selects the appropriate parking area(s).

← 4. **System** prompts for pricing options including recurring reservation parking rate, confirmed reservation parking rate, penalty fees, etc.

→ 5. **System Admin** enters in the desired prices.

← 6. **System** prompts the user to confirm the desired changes.

→ 7. **System Admin** selects “Yes, confirm the pricing changes”.

← 8. **System** updates the **Database** to include the new pricing options for the parking area.

Flow of Events for Extensions (Alternate Scenario):

7a. **System Admin** selects “No, I have made a mistake”.

Use Case UC-12:	**Inspect Usage History** (sub-use case)

Related Requirements:	REQ13
Initiating Actor:	System Admin
Actor’s Goal:	To examine the usage history of a particular parking
Participating Actors:	area. Database
Preconditions:	System Admin is currently logged into the system.

	include::AuthenticateUser (UC-9)
Postconditions:	None

Flow of Events for Main Success Scenario:

→ 1. **System Admin** selects the menu option “Inspect Usage History”.

← 2. **System** prompts for search criteria including parking area location, start date, and end date.

→ 3. **System Admin** specifies the search criteria and submits.

← 4. **System** prepares a database query that best matches the actor’s search criteria and retrieves the records from the **Database**.

→ 5. **Database** returns the matching records.

← 6. **System** (a) additionally filters the retrieved records to match the actor’s search criteria, (b) renders the remaining records for display, and (c) shows the result for the **System Admin** to view.

Use Case UC-13:	**MonthlyBilling**

Related Requirements:	REQ11, REQ12
Initiating Actor:	Timer
Actor’s Goal:	To generate and send a monthly bill to every registered person in the database.
Participating Actors:	Database
Preconditions:	It is the last day of the month.
Postconditions:	None

Flow of Events for Main Success Scenario:

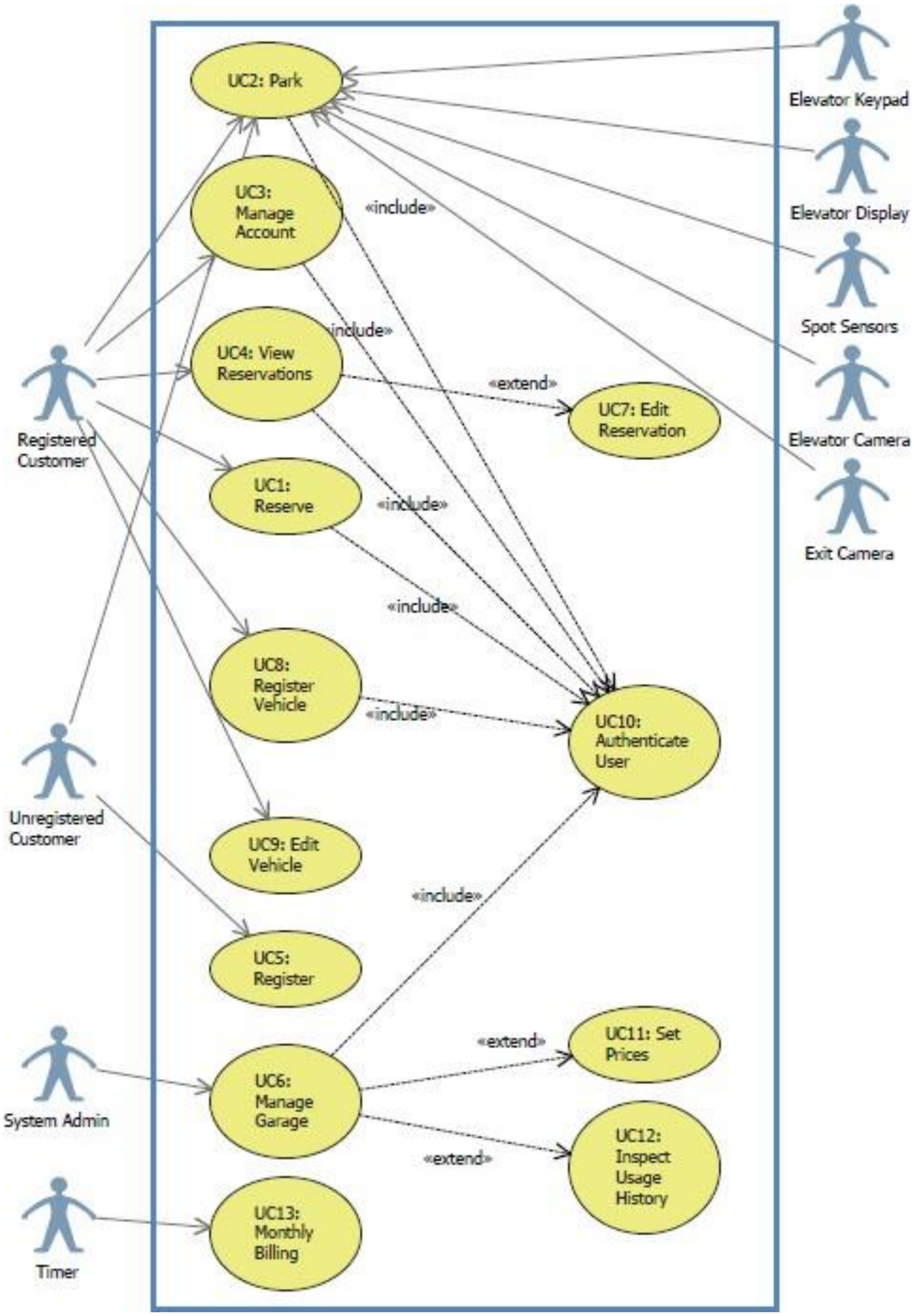
→ 1. **Timer** notifies the **System** that it is the last day of the month

← 2. **System** queries the **Database** for all registered person IDs.

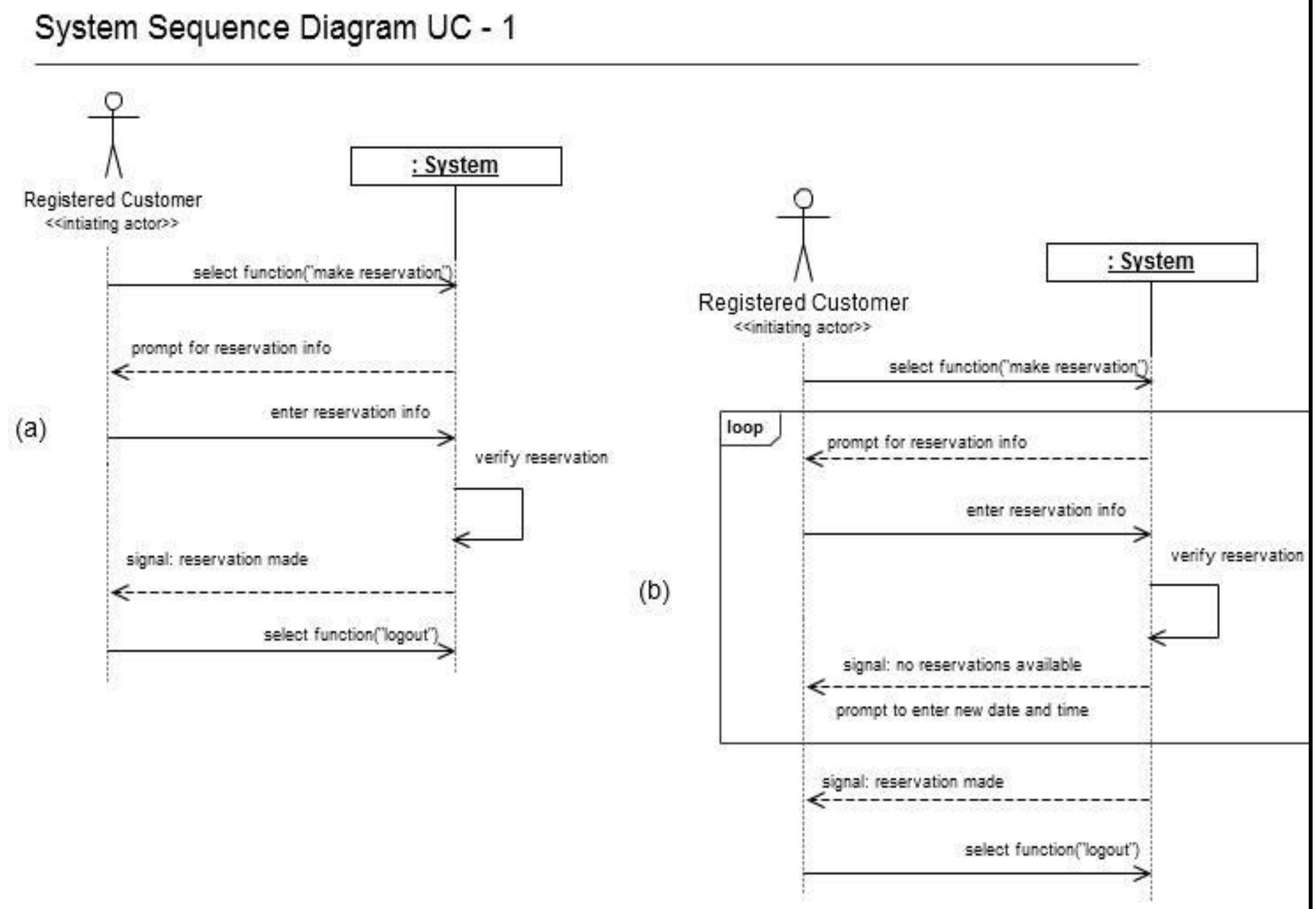
← 3. **System** queries the **database** for all accumulated parking hours and fees for each registered person.

← 4. **System** (a) calculates the total charges (b) generates a bill from the calculations, and © emails one to each registered person.

Use Case Diagram

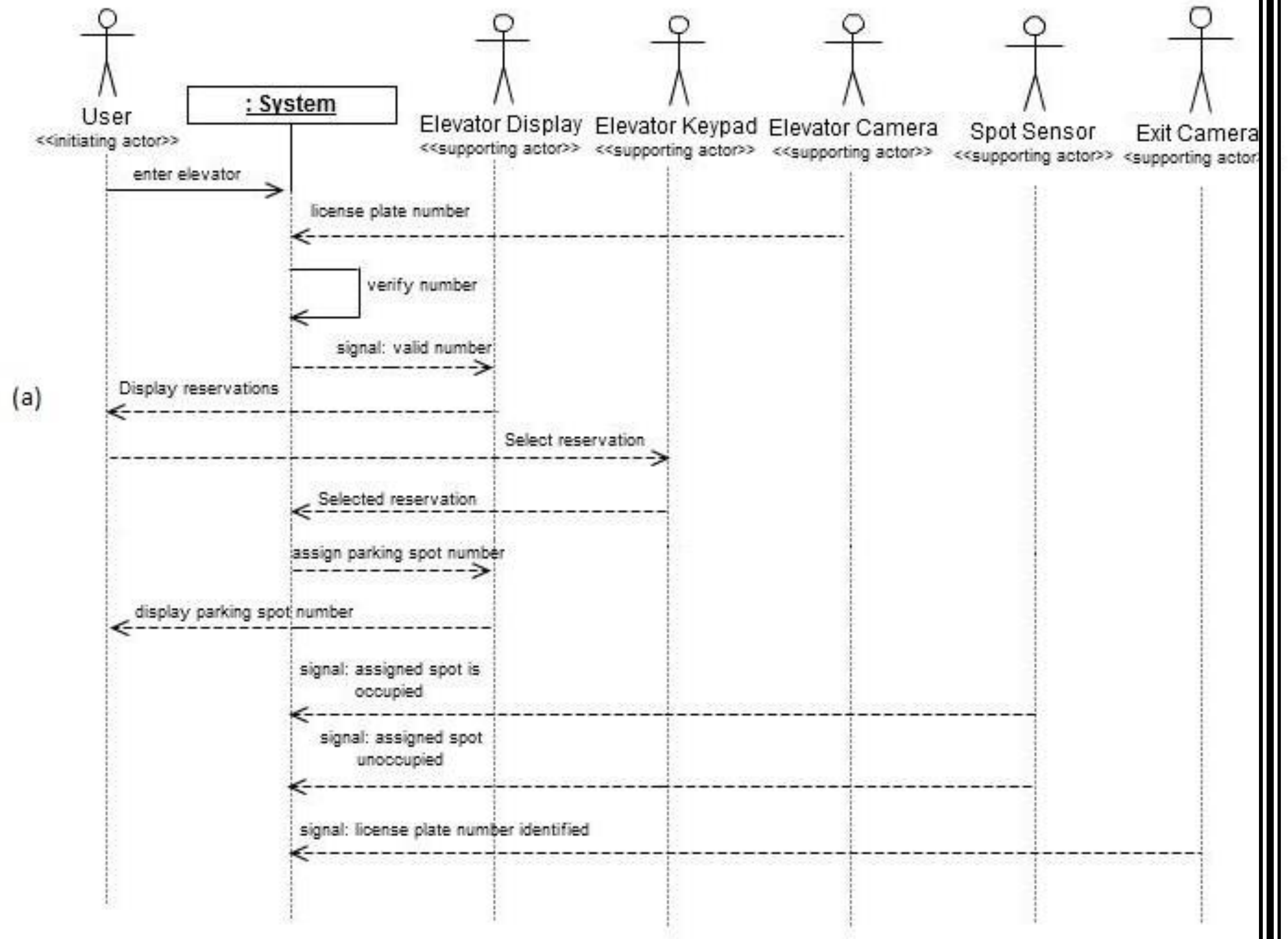


System Sequence Diagrams

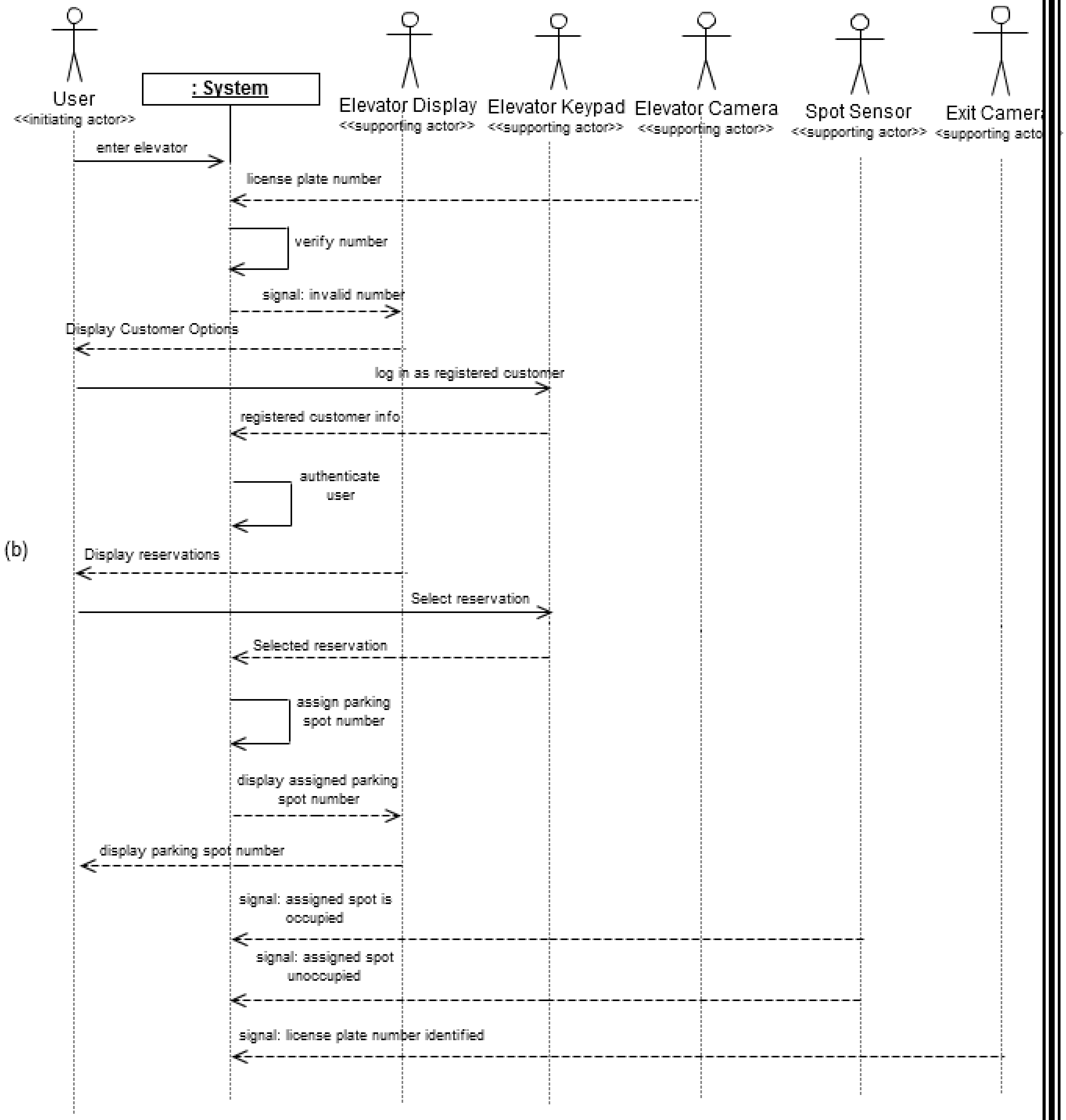

System Sequence Diagram UC - 1
Registered Customer
<<intiating actor>>
: System
select function("make reservation")
prompt for reservation info
(a)
enter reservation info
verify reservation
signal: reservation made
select function("logout")
Registered Customer
<<initiating actor>>
: System
select function("make reservation")
loop
prompt for reservation info
enter reservation info
verify reservation
(b)
signal: no reservations available
prompt to enter new date and time
signal: reservation made
select function("logout")

The System sequence diagram for UC-1 can be split up into two parts. Part (a) describes the sequence of events for the success scenario. The user requests a reservation and there are available reservations to be given out. Part b describes the sequence of events for the alternate scenario. The user requests a reservation at a specific date and time but the there are no available reservations to be given out. The user will continue to enter in a new date and time until an available reservation can be given out.

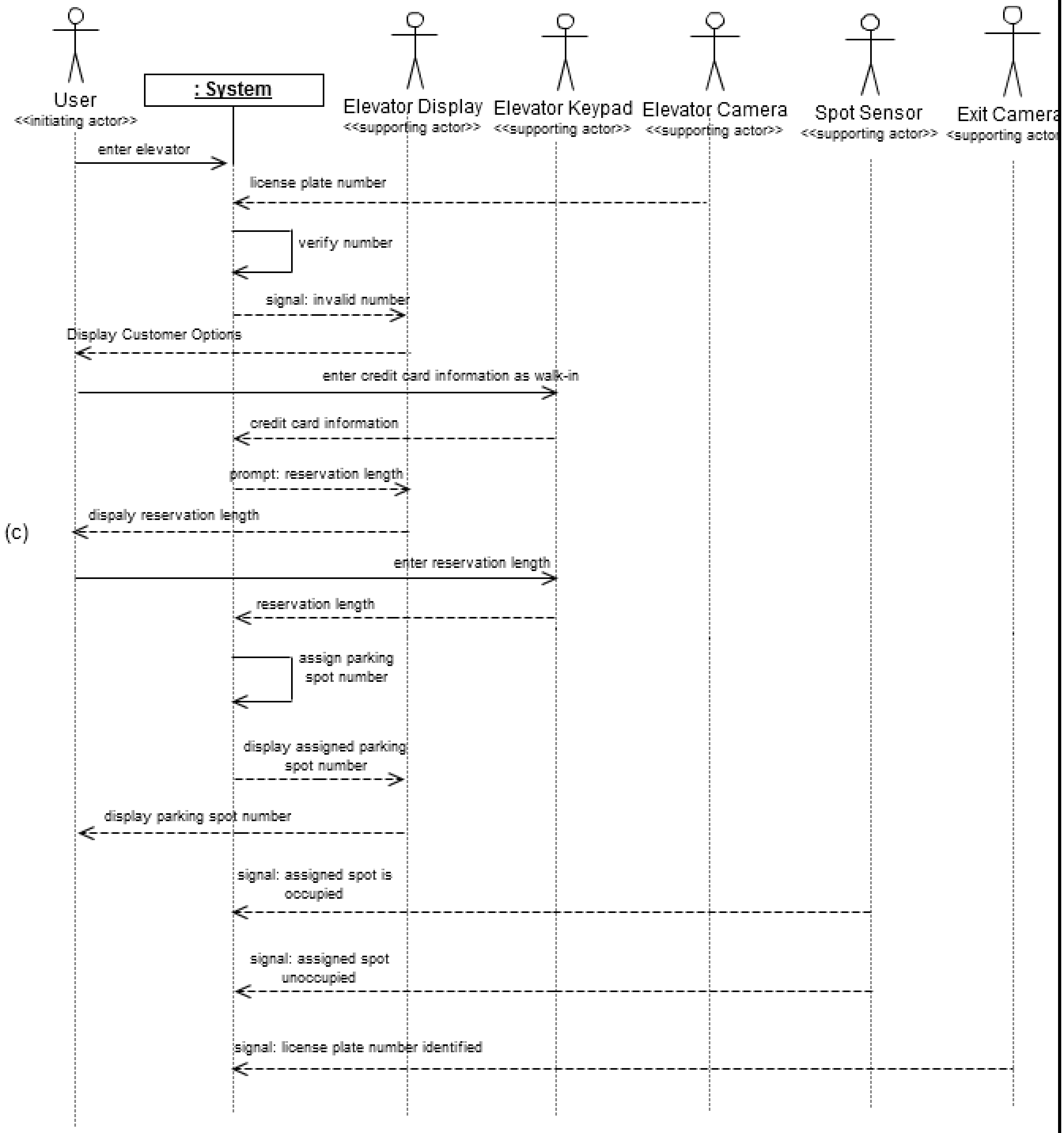
Sequence Diagram UC - 2



The system sequence diagram for UC-2 can be split into three parts. Part a describes the sequence of events for the success scenario. The person enters the elevator, the elevator camera recognizes the the person’s license plate number, and the system assigns the user a parking spot number.



Part (b) of the system sequence diagram for UC-2 describes the sequence of events for an alternate scenario. The person enters the elevator but the elevator camera does not recognize the person's license plate number. The person then signs in as a registered person and selects a reservation. The system then assigns the user a parking spot number.



Part (c) of the system sequence diagram for UC-2 describes the sequence of events for another alternate scenario. The person enters the elevator but the elevator camera does not recognize the person's license plate number. The person then swipes their credit card and selects a reservation as a walk-in person. The system then assigns the user a parking spot number.

Non-Functional Requirements

Fault-tolerance

- Park-A-Lot should remember the details of a user’s interaction if the user interface should disconnect from the system.
- Park-A-Lot should quickly recover from a malfunction when a person is inside the elevator.

Usability

- The interface should provide persons with access to all relevant use cases with the fewest number of mouse clicks and key strokes.

Reliability

- Park-A-Lot should function correctly even if a person inputs invalid entries into a reservation request form.
- Park-A-Lot should not lose a reservation through the use of persistent storage and regular backup.

Performance

- The Park-A-Lot elevator display should always display the correct output to the person.
- The Park-A-Lot system should minimize connection times to the database and provide a quick and painless experience to the person.
- Initially, Park-A-Lot can support at least 100 persons and 1,000 reservations. Over time should seek to increase these numbers ten-fold or more.

Security

- Other persons or unauthorized users should not have access to or be able to edit a person’s account details or reservations.

Effort Estimation Using Use Case Points

Unadjusted Actor Weight (UAW)

Actor Name	Description of relevant characteristics	Complexity	Weight
Registered Person	Registered Person is interacting with the system via the website or with the elevator keypad and display.	Complex	3
Unregistered Person	Unregistered Person is interacting with the system via the website (to create an account) or with the elevator keypad and display.	Complex	3
System Admin	The System Admin is interacting with the system via the website (to set prices and view access history).	Complex	3
Timer	Timer is another system which interacts with our system through a defined API.	Simple	1
Elevator Camera	Same as Timer.	Simple	1
Database	Database is another system interacting through a protocol.	Average	2
Spot Sensor	Same as Timer.	Simple	1
Elevator Keypad	Same as Timer.	Simple	1
Elevator Display	Same as Timer.	Simple	1
Exit Camera	Same as Timer.	Simple	1

UAW = 6 x Simple + 1 x Average + 3 x Complex = 6x1 + 1x2 + 3x3 = 17

Unadjusted Use Case Weight (UUCW)

Use Case	Description	Category	Weight
Reserve UC-1	Simple user interface. 3 steps for the main success scenario. 1 participating actor (Database).	Simple	5
Park UC-2	Complex user interface. More than 7 steps for all the scenarios. 6 participating actors (Elevator Display, Elevator Camera, Elevator Keypad, Spot Sensor, Exit Camera, Database).	Complex	10
Manage Account UC-3	Simple user interface. 4 steps for the main success scenario. 1 participating actor (Database).	Simple	5
View Reservations UC-4	Complex user interface. 4 steps for the main success scenario. 1 participating actor (Database).	Average	10
Register UC-5	Simple user interface. More than 7 steps for all the scenarios. 1 participating actor (Database).	Average	10
Manage Parking area UC-6	Simple user interface. 6 steps required for the main success scenario. 1 participating actor (Database).	Average	10
Edit Reservation UC-7	Simple user interface. 4 steps required for the main success scenario. 1 participating actor (Database).	Simple	5
Register Vehicle UC-8	Simple user interface. 4 steps required for the main success scenario. 1 participating actor (Database).	Simple	5
Edit Vehicle UC-9	Simple user interface. 4 steps required for the main success scenario. 1 participating actor (Database).	Simple	5
Authenticate User UC-10	Simple user interface. 7 steps for all the scenarios. 1 participating actor (Database).	Average	10
Set Prices UC-11	Simple user interface. 8 steps for the main success scenario. 1 participating actor (Database).	Average	10

Inspect Usage History UC-12	Complex user interface. 6 steps for the main success scenario. 1 participating actor (Database).	Complex	15
Monthly Billing UC-13	Simple user interface. 4 steps for the main success scenario. 1 participating actor (Database).	Simple	5

UUCW = 6 x Simple + 5 x Average + 2 x Complex = 6x5 + 5x10 + 2x15 = 110

UUCP = UAW + UUCW = 17 + 110 = 127

Technical Complexity Factor (TCF)

Technical Factor	**Description**	**Weight**	**Perceived Complexity**	**Calculated Factor (Weight x Perceived Complexity)**
T1	Distributed, Web-based system.	2	5	2x5 = 10
T2	User expect good performance and no down times.	1	3	1x3 = 3
T3	End-users expect efficiency.	1	3	1x3 = 3
T4	Internal processing is relatively simple except reservation swapping (UC-1).	1	4	1x4 = 4
T5	Reusability is a must have feature.	1	4	1x4 = 4
T6	Ease of install is not important because its a web based system.	0.5	0	0.5x0 = 0
T7	Ease of use is very important.	0.5	3	0.5x3 = 1.5
T8	Portability could be important for future improvements (phone app).	2	0	2x0 = 0
T9	Ease to change is required.	1	3	1x3 = 3

T10	Concurrent use is required.	1	3	1x3 = 3
T11	Security is a significant concern.	1	4	1x4 = 4
T12	No direct access for third parties.	1	0	1x0 = 0
T13	No unique training needs.	1	0	1x0 = 0

TCF = Constant-1 + Constant-2 x Technical Factor Total =
0.6 + 0.01 x (10 + 3 + 3 + 4 + 4 + 1.5 + 3 + 3 + 4) = 0.955

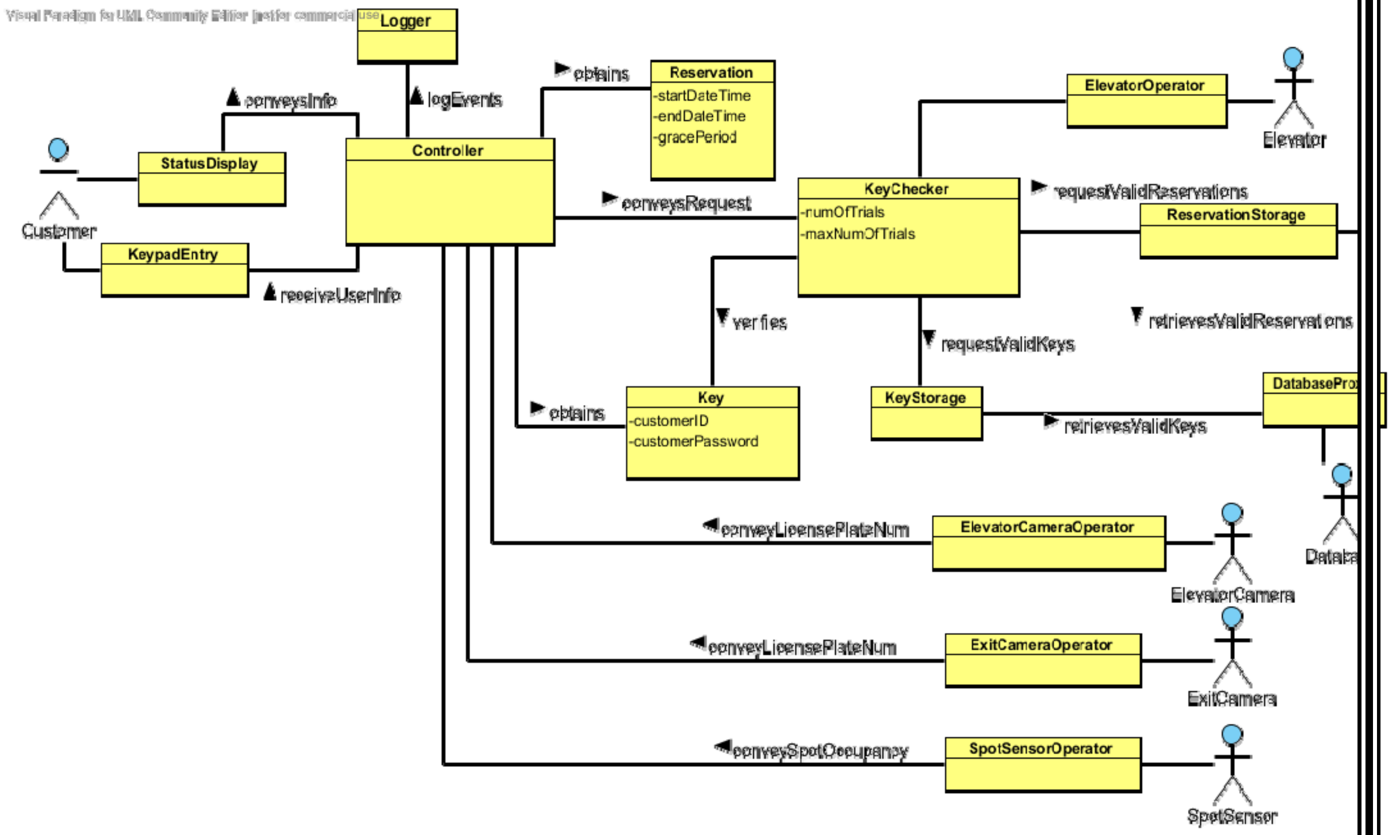
Environmental Complexity Factor (ECF)

Environmental Factor	**Description**	**Weight**	**Perceived Impact**	**Calculated Factor (Weight x Perceived Impact)**
E1	Beginner familiarity with the UML- based development.	1.5	3	1.5 x 3 = 4.5
E2	Some familiarity with application problem.	0.5	3	0.5 x 3 = 1.5
E3	Some knowledge of object-oriented approach.	1	2	1 x 2 = 2
E4	Some knowledge of lead analyst.	0.5	2	0.5 x 2 = 1
E5	Highly motivated but lost one team member.	1	4	1 x 4 = 4
E6	Stable requirements expected.	2	1	2 x 1 = 2
E7	All staff is part-time (have other class work to do as well).	-1	4	-1 x 4 = -4
E8	Programming language of average difficulty will be used.	-1	3	-1 x 3 = -3

Domain Analysis

Domain Models

Domain Model for UC-2



Reasons for model selection:

- **Cohesion** - The domain elements contain enough information to be completely independent objects (no overlap in knowledge between objects) and represent concrete ideas within the system. Therefore, our model has high cohesion since each object has several responsibilities, but does not attempt to do too much work.
- **Coupling** - The coupling of objects in our diagram is low, mainly because we have separated the key checking ability out from the Controller object. Key checking is a main concern in this model, and therefore deserves it's own object. The smaller objects surrounding KeyChecker help it accomplish its job. There is a high degree of coupling between Controller and many of the physical objects in the parking parking area, but that is impossible to avoid since the controller needs to be in communication with all of the cameras and sensors in order to instruct other objects when to complete their tasks.
- **Expert Doer Principle** - The model satisfies this principle because it divides the task of checking keys and processing information about those checked keys into two distinct objects. The KeyChecker is the expert on checking person authentication information,

and informs the Controller of person authenticity. The Controller is then able to quickly display information about Reservations to the person.

(D - doing; K - knowing; N - neither)

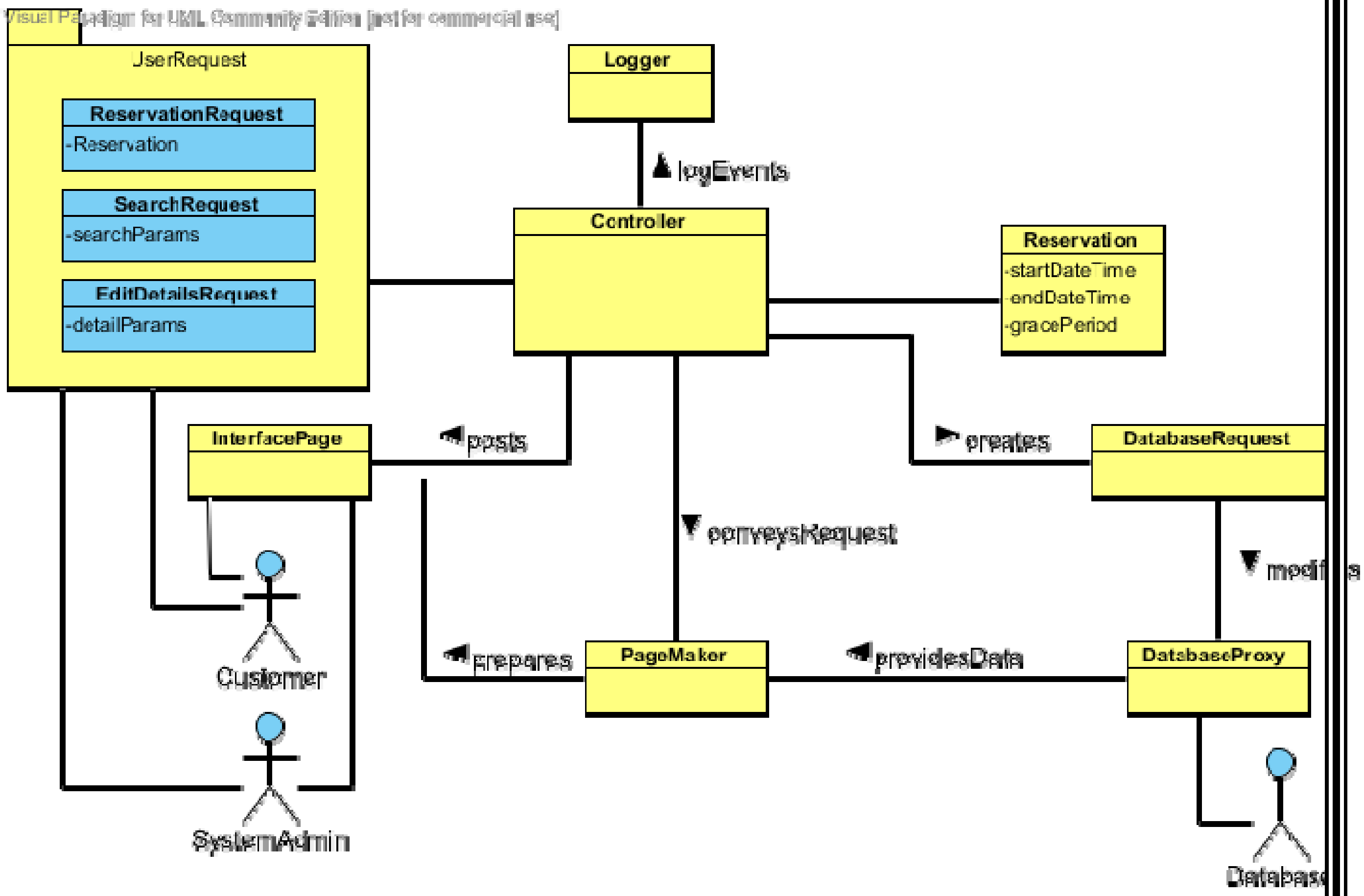
Responsibility Description	**Type**	**Concept Name**
Coordinate actions of concepts associated with this use case and delegate the work to other concepts.	D	Controller
Shows the actor the current context, what actions can be done, and the outcomes of the previous actions.	N	StatusDisplay
Container for the person ID and password that the user entered in.	N	KeypadEntry
Container for user's authentication data, including Person ID and password.	K	Key
Verify whether or not the key-code entered by the user is valid.	D	KeyChecker
Container for the collection of valid keys associated with the users.	K	KeyStorage
Container for user's existing reservations.	K	Reservation
Container for the collection of reservations associated with each user.	D	ReservationStorage
Operate the elevator camera to identify a car's license plate number in the elevator platform.	D	ElevatorCameraOperator
Operate the elevator to move to the correct floor and open the entrance door.	D	ElevatorOperator
Operate the spot sensor to determine if a car is parked in the parking spot.	D	SpotSensorOperator
Operate the exit camera to identify the car's license plate number that is exiting the parking area.	D	ExitCameraOperator
Log all interactions with the system in persistent storage.	D	Logger

Concept Pair	**Association Description**	**Association Name**
Controller ↔ StatusDisplay	Controller passes information concerning the current context, what actions can be done, and the outcomes of the previous actions	conveysInfo
Controller ↔ KeypadEntry	Controller receives user input information from KeypadEntry.	receiveUserInfo
Controller ↔ Logger	Controller logs information to persistent storage about system interactions.	logEvents
Controller ↔ Reservation	Controller obtains reservation information from Reservation container.	obtains
Controller ↔ KeyChecker	Controller requests KeyChecker validate person ID/password input and receives any reservations that ID is linked to.	conveysRequest
Controller ↔ Key	Controller obtains verified user information from Key container.	obtains
ElevatorCameraOperator ↔ Controller	ElevatorCameraOperator conveys available license plate number to Controller when ElevatorCamera detects a car in the Elevator.	conveyLicensePlateNum
ExitCamera Operator ↔ Controller	ExitCameraOperator conveys available license plate number to Controller when ExitCamera detects a car leaving the parking parking area.	conveyLicensePlateNum
SpotSensor Operator ↔ Controller	SpotSensorOperator conveys true/false if a spot is filled to Controller.	conveySpotOccupancy
KeyChecker ↔ Key	KeyChecker verifies if Key matches given user account information.	verifies
KeyChecker ↔ KeyStorage	KeyChecker requests a list of valid keys from KeyStorage container.	requestValidKeys

KeyChecker ↔ Reservation Storage	KeyChecker retrieves a list if reservations from ReservationStorage container.	requstValid Reservations
KeyChecker ↔ Elevator Operator	KeyChecker tells the ElevatorOperating what floor to go to.	signalOperateElevator
Reservation Storage ↔ DatabaseProxy	ReservationStorage queries DatabaseProxy for valid reservations.	retrievesValid Reservations
KeyStorage ↔ DatabaseProxy	KeyStorage queries DatabaseProxy for valid keys.	retrievesValidKeys

Concept	**Attributes**	**Attribute Description**
Reservation	Start Date Time End Date Time Grace Period	Start time for the reservation of a particular actor. End time for the reservation of a particular actor. Grace period for the reservation of a particular actor.
Key	Person’s ID Person’s Password	ID number of person. Password for the person.
Key Checker	Number of Trials Max Number of Trials	Counter to track how many times the user has unsuccessfully entered in their key. Maximum allowable times a user can enter in their key unsuccessfully before they are asked to leave.

Domain Model for Remaining Use Cases



Reasons for model selection:

- **Cohesion** - The responsibilities assigned to each object in this model are not as great as they first appear, since the UserRequest object is actually a collection of three distinct objects which are used depending upon person or system administrator input. The ReservationRequest is used in the case that the person seeks to create a reservation, the SearchRequest in the case the person or system administrator is searching for a set of reservations in the database, and the EditDetailsRequest in the case where a person needs to edit some info in his or her account, or a system administrator needs to edit the details of an account. This gives each object a clearly defined set of responsibilities. The remaining objects are simply there to create and display the data fetched from the database, and each have a unique function, thus promoting high cohesion.
- **Coupling** - While there are many responsibilities assigned to each object, no object has an excessive amount of work to do. Each object has a distinct task to perform, and the Controller oversees these tasks and coordinates their efforts. The coupling is lower in this domain model than in the previous one, since we have reduced the interactions between Controller and all other objects, while still maintaining a distinct set of tasks for each object.

Interaction Diagrams

Theoretical Interaction Diagrams

For UC-2, when the person enters the parking parking area, we have created a few interaction diagrams that would be useful in an actual commercial implementation but not for our demonstration purposes. We will not be demonstrating these interaction diagrams during the demo, simply because we cannot afford to build a parking area with all of the required hardware to actually implement the signals our system would be receiving. However, we will include a way to simulate this in the coding of our final project.

The first diagram demonstrates the decisions to be made when a person enters the parking area elevator. If the person is Registered and has an existing reservation, then they are called to UC-2 Registered Person. If the person is Registered but does not have an existing reservation then they are either called to UC-2 Registered Person or UC-2 Walk In. If the person is unregistered then they are called to UC-2 Walk In.

The interaction diagram entitled UC-2 Registered Person walks through a Registered Person logging into the keypad with their password and then selecting which reservation they wish to fulfill. If the person does not have an existing reservation then they will create one if it is available. The person will then be called to UC-2 Park.

The interaction diagram entitled UC-2 Walk In, walks through a Registered or Unregistered person reserving a spot (if one is available) and then proceeding to UC-2 Park. This interaction diagram utilizes UC-1 Reserve.

The interaction Diagram entitled UC-2 Park, walks through the person going to the correct floor, parking in the assigned parking spot and then leaving when the reservation is over. The interaction diagram utilized the Elevator Operator, the Spot Sensor Operator, the Elevator Camera Operator, and the Exit Camera Operator. These concepts are used to move the elevator to the correct floor, identify if a specific parking spot is full, recognize a car’s license plate number, and notify the system when the person has left the parking area. In our demonstration, we cannot utilize these concepts, so we have implemented them by using user interaction. The demo interaction diagrams will be explained in detail in the next section on implemented Interaction Diagrams.

Implemented Interaction Diagrams

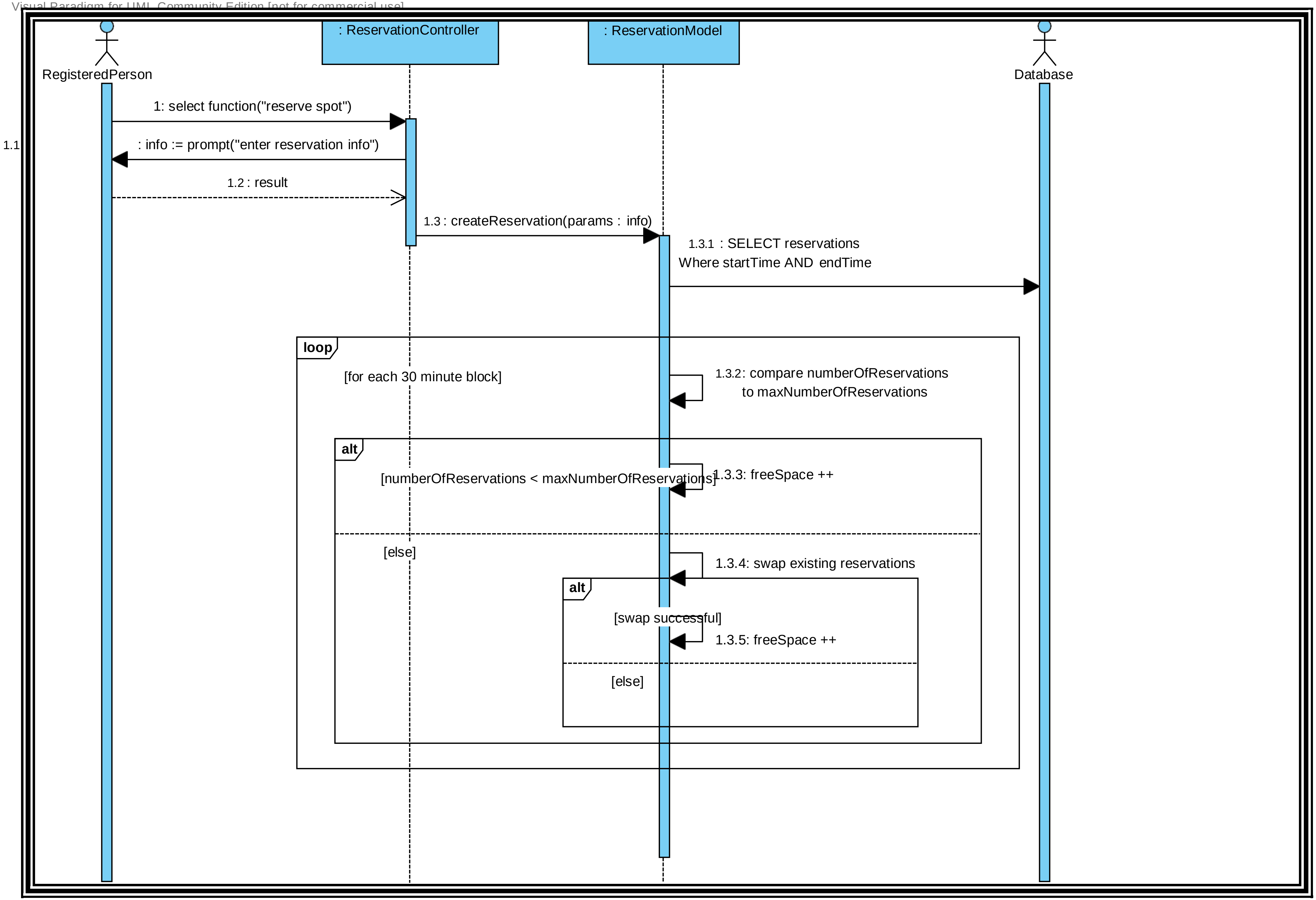
All of the interaction diagrams demonstrate our use of the MVC (model, view, controller) framework. By using this type of framework, we can separate business logic from the controller and from the views. Within our architecture we have 5 different MVC groups listed below.

- User
- Reservation
- Vehicle
- Admin
- Timer

UC-1: Reserve

Goals: To create a reservation and add it to the Database.

Process: The person is prompted to input information about the reservation they wish to create, then the Controller passes the information to the Model, which determines if the reservation is valid and does not overbook the parking area. The person is then given either a success page or asked to re-enter the information with different values.


RegisteredPerson
: ReservationController
: ReservationModel
Database
1: select function("reserve spot")
1.1 : info := prompt("enter reservation info")
1.2 : result
1.3 : createReservation(params : info)
1.3.1 : SELECT reservations
Where startTime AND endTime
loop
[for each 30 minute block]
1.3.2: compare numberOfReservations
to maxNumberOfReservations
alt
[numberOfReservations < maxNumberOfReservations]
1.3.3: freeSpace ++
[else]
1.3.4: swap existing reservations
alt
[swap successful]
1.3.5: freeSpace ++
[else]

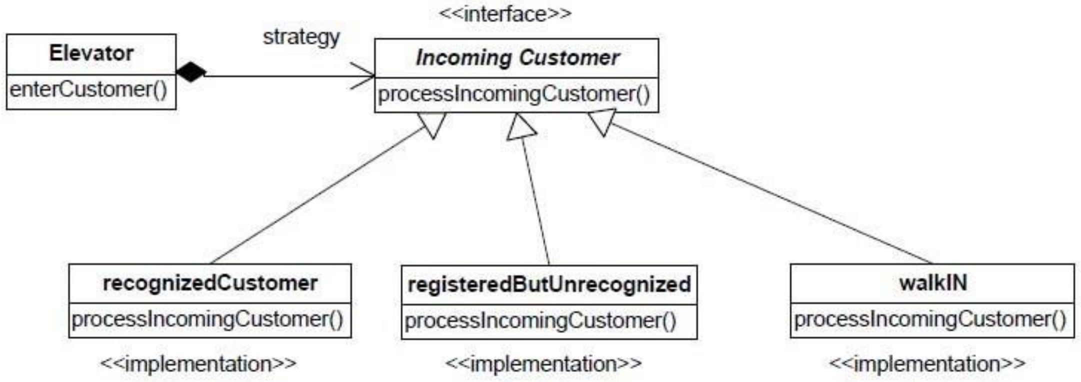
UC-2: Park

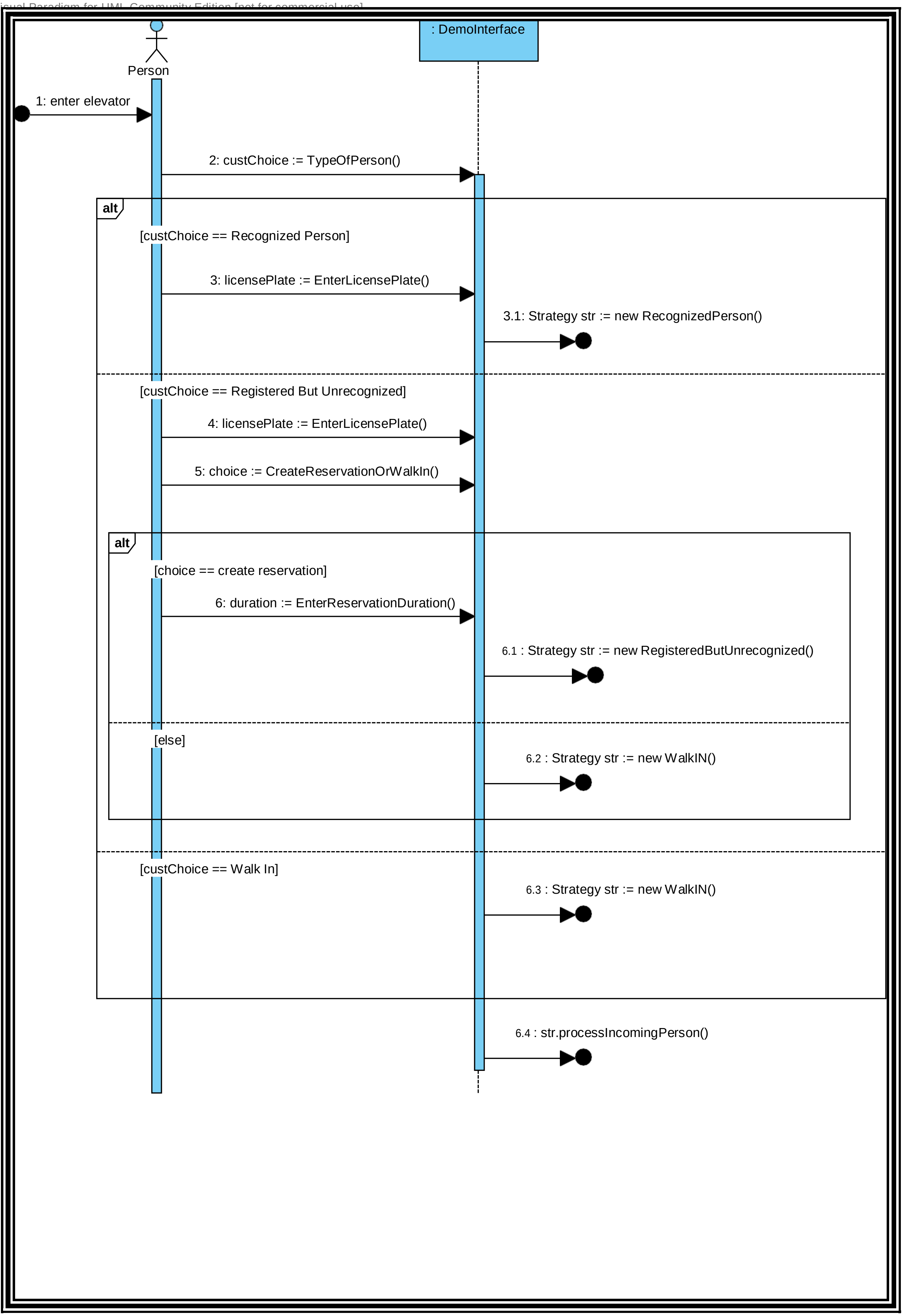
Goals: To park in the parking area, either to fulfill a reservation or as a walk in.

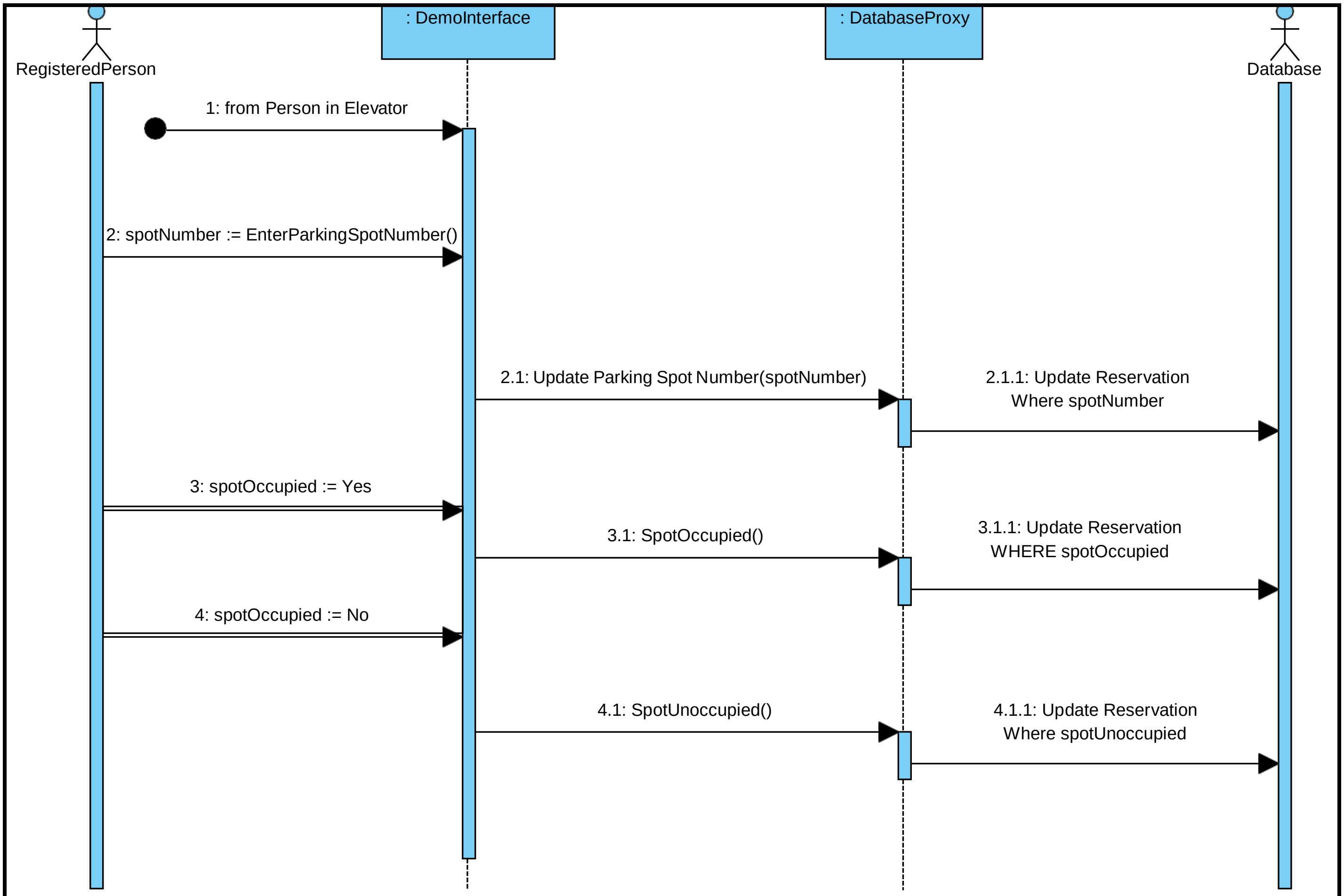
Process: A person enters the parking area and is prompted to select one of three options; Registered Person with a reservation, Registered Person without a reservation, or Walk In. Based on the selection the person will be required to enter in his/her license plate number and then be told in what parking spot to park. The person then has the option to leave the parking area and is informed of how much the parked cost. This implementation is only for simulation purposes and is not intended to be used for an actual parking parking area.

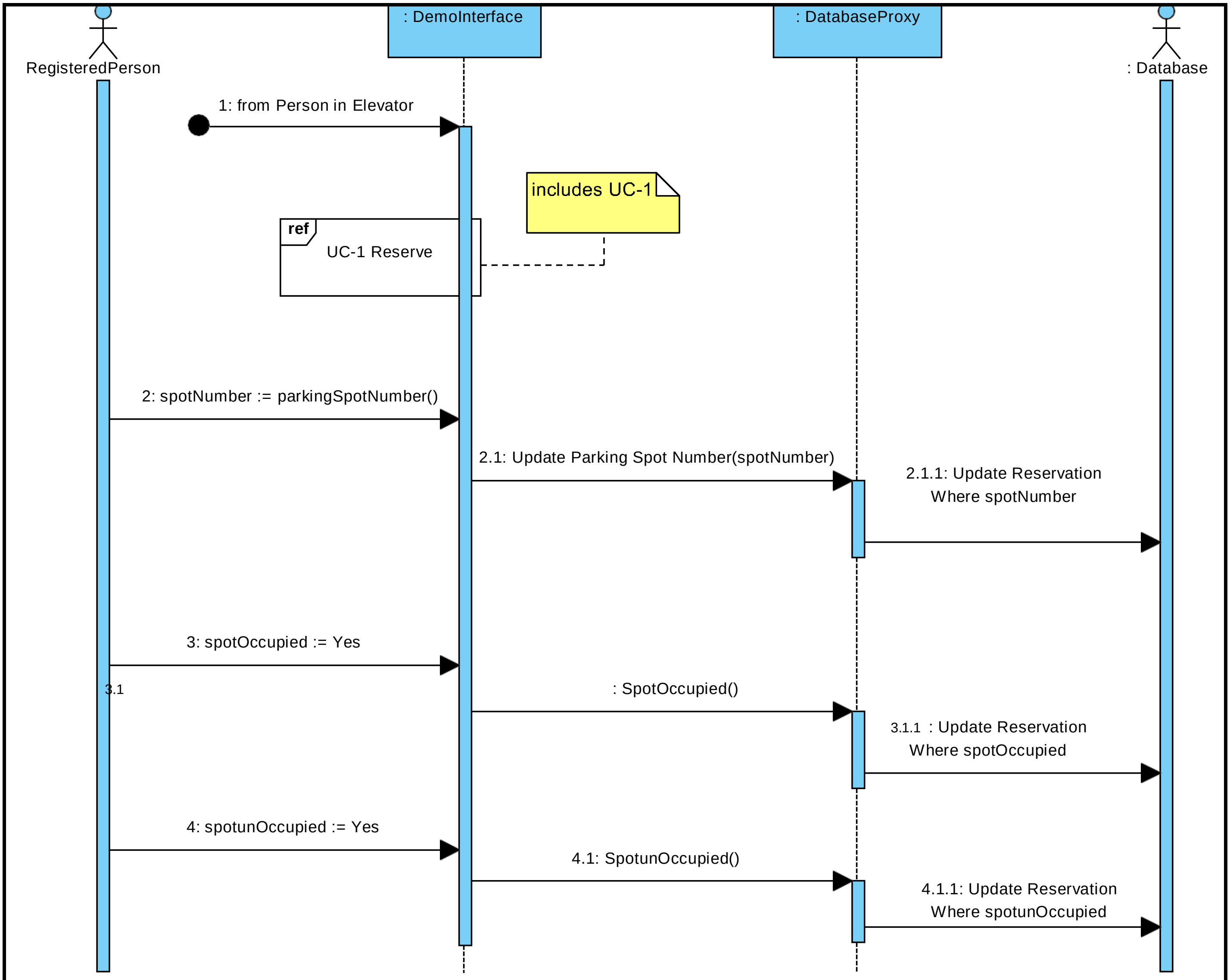
Design Pattern:
We have employed the strategy pattern in use case 2, Park. We decided to use this pattern because UC-2 has three different algorithms / options pertaining to the user's input and by employing this pattern we can simplify the logic involved. All three of the algorithms / options are chosen at run time so this patter was an obvious choice. The three algorithms we have employed are Recognized Person, Registered But Unrecognized, and Walk In. All three strategies are inherited by the incoming person interface. The diagram below shows the strategies and their inherited methods.

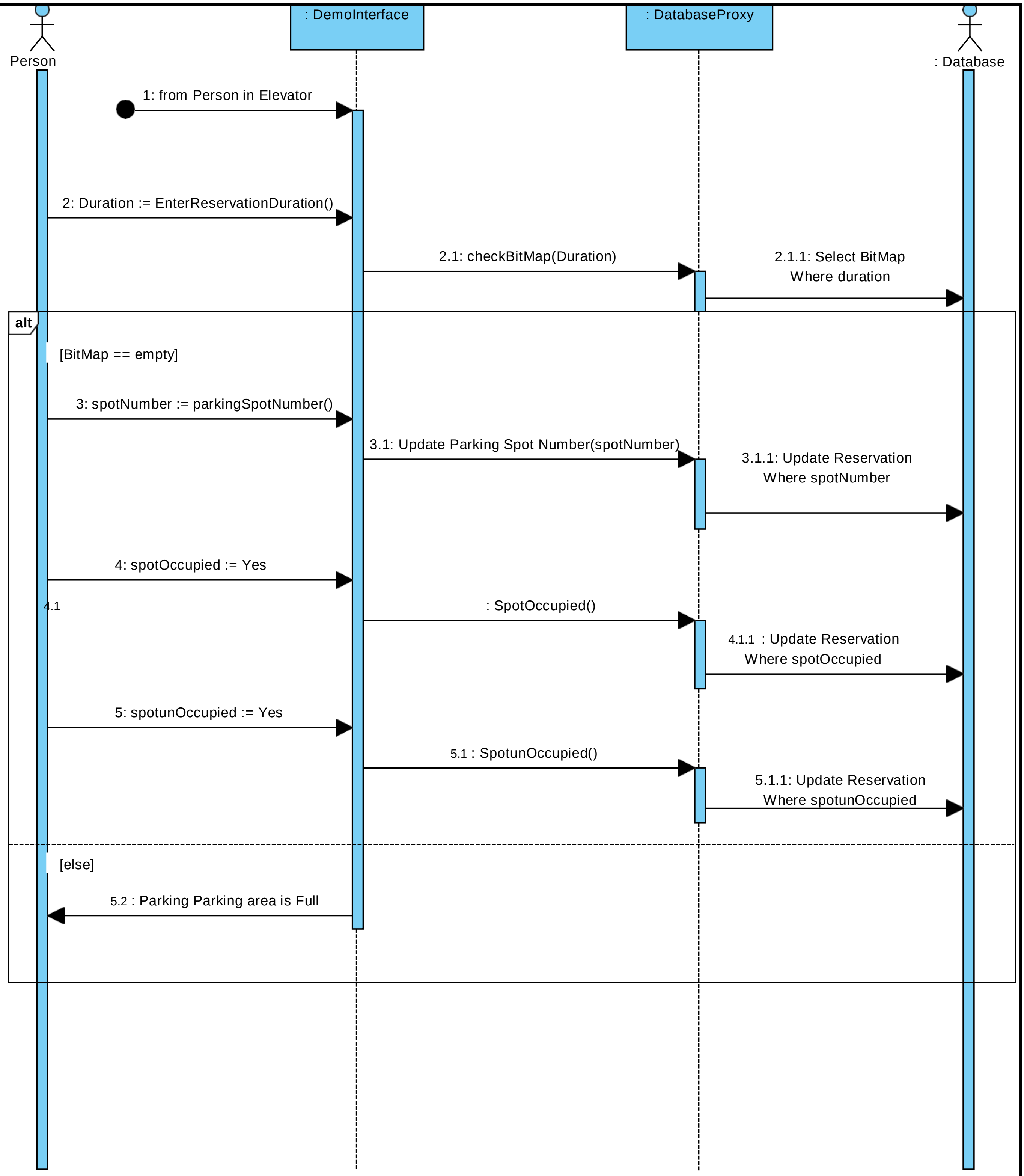
The advantaged to using the strategy pattern are that the strategies are encapsulated as an object and then made interchangeable. It would be easy in the future to add, remove or change any of the strategies because they are each a separate object called from their parent interface. Also, since all of the information relating to picking which strategy to use is entered during run time, the strategy patterns limits the amount of hard coding that is necessary to utilize each option. Each strategy is stored as a reference to the actual object which makes it easy to create and destroy.




Person
: DemoInterface
1: enter elevator
2: custChoice := TypeOfPerson()
alt
[custChoice == Recognized Person]
3: licensePlate := EnterLicensePlate()
3.1: Strategy str := new RecognizedPerson()
[custChoice == Registered But Unrecognized]
4: licensePlate := EnterLicensePlate()
5: choice := CreateReservationOrWalkIn()
alt
[choice == create reservation]
6: duration := EnterReservationDuration()
6.1 : Strategy str := new RegisteredButUnrecognized()
[else]
6.2 : Strategy str := new WalkIN()
[custChoice == Walk In]
6.3 : Strategy str := new WalkIN()
6.4 : str.processIncomingPerson()


RegisteredPerson
: DemoInterface
: DatabaseProxy
Database
1: from Person in Elevator
2: spotNumber := EnterParkingSpotNumber()
2.1: Update Parking Spot Number(spotNumber)
2.1.1: Update Reservation
Where spotNumber
3: spotOccupied := Yes
3.1: SpotOccupied()
3.1.1: Update Reservation
WHERE spotOccupied
4: spotOccupied := No
4.1: SpotUnoccupied()
4.1.1: Update Reservation
Where spotUnoccupied

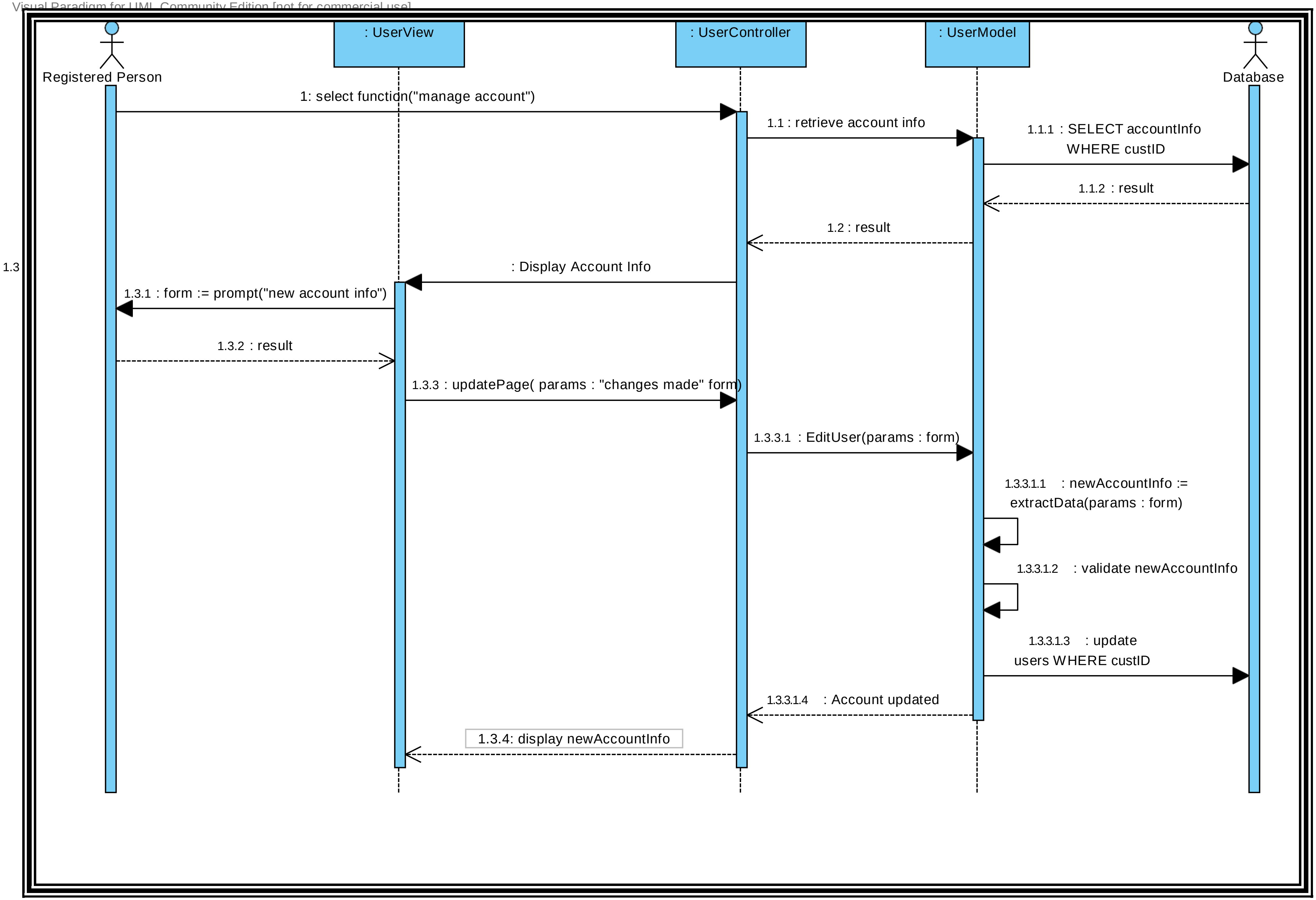

: DemoInterface
: DatabaseProxy
RegisteredPerson
: Database
1: from Person in Elevator
includes UC-1
ref
UC-1 Reserve
2: spotNumber := parkingSpotNumber()
2.1: Update Parking Spot Number(spotNumber)
2.1.1: Update Reservation
Where spotNumber
3: spotOccupied := Yes
3.1
: SpotOccupied()
3.1.1 : Update Reservation
Where spotOccupied
4: spotunOccupied := Yes
4.1: SpotunOccupied()
4.1.1: Update Reservation
Where spotunOccupied


: DemoInterface
: DatabaseProxy
Person
: Database
1: from Person in Elevator
2: Duration := EnterReservationDuration()
2.1: checkBitMap(Duration)
2.1.1: Select BitMap
Where duration
alt
[BitMap == empty]
3: spotNumber := parkingSpotNumber()
3.1: Update Parking Spot Number(spotNumber)
3.1.1: Update Reservation
Where spotNumber
4: spotOccupied := Yes
4.1
: SpotOccupied()
4.1.1 : Update Reservation
Where spotOccupied
5: spotunOccupied := Yes
5.1 : SpotunOccupied()
5.1.1: Update Reservation
Where spotunOccupied
[else]
5.2 : Parking Parking area is Full

UC-3: Manage Account

Goals: To change account details for a registered person.

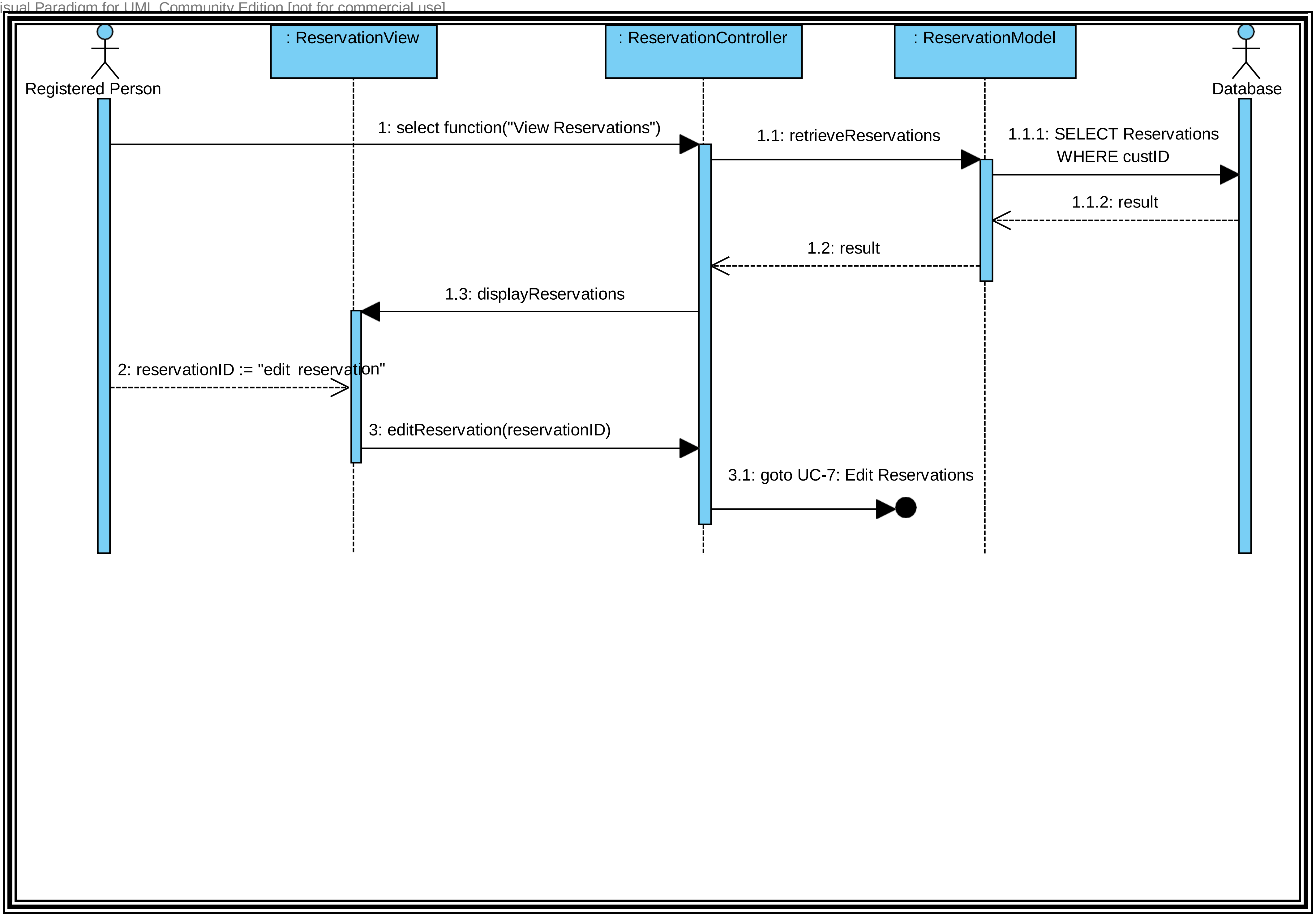
Process: The registered person is prompted to make changes to their current account information. The controller passes the information to the Model, which updates the account in the database and then displays a page signifying the successful update.


Registered Person
: UserView
: UserController
: UserModel
Database
1: select function("manage account")
1.1 : retrieve account info
1.1.1 : SELECT accountInfo WHERE custID
1.1.2 : result
1.2 : result
1.3
: Display Account Info
1.3.1 : form := prompt("new account info")
1.3.2 : result
1.3.3 : updatePage(params : "changes made" form)
1.3.3.1 : EditUser(params : form)
1.3.3.1.1 : newAccountInfo := extractData(params : form)
1.3.3.1.2 : validate newAccountInfo
1.3.3.1.3 : update users WHERE custID
1.3.3.1.4 : Account updated
1.3.4: display newAccountInfo

UC-4: View Reservations

Goals: To view existing reservations for a registered person.

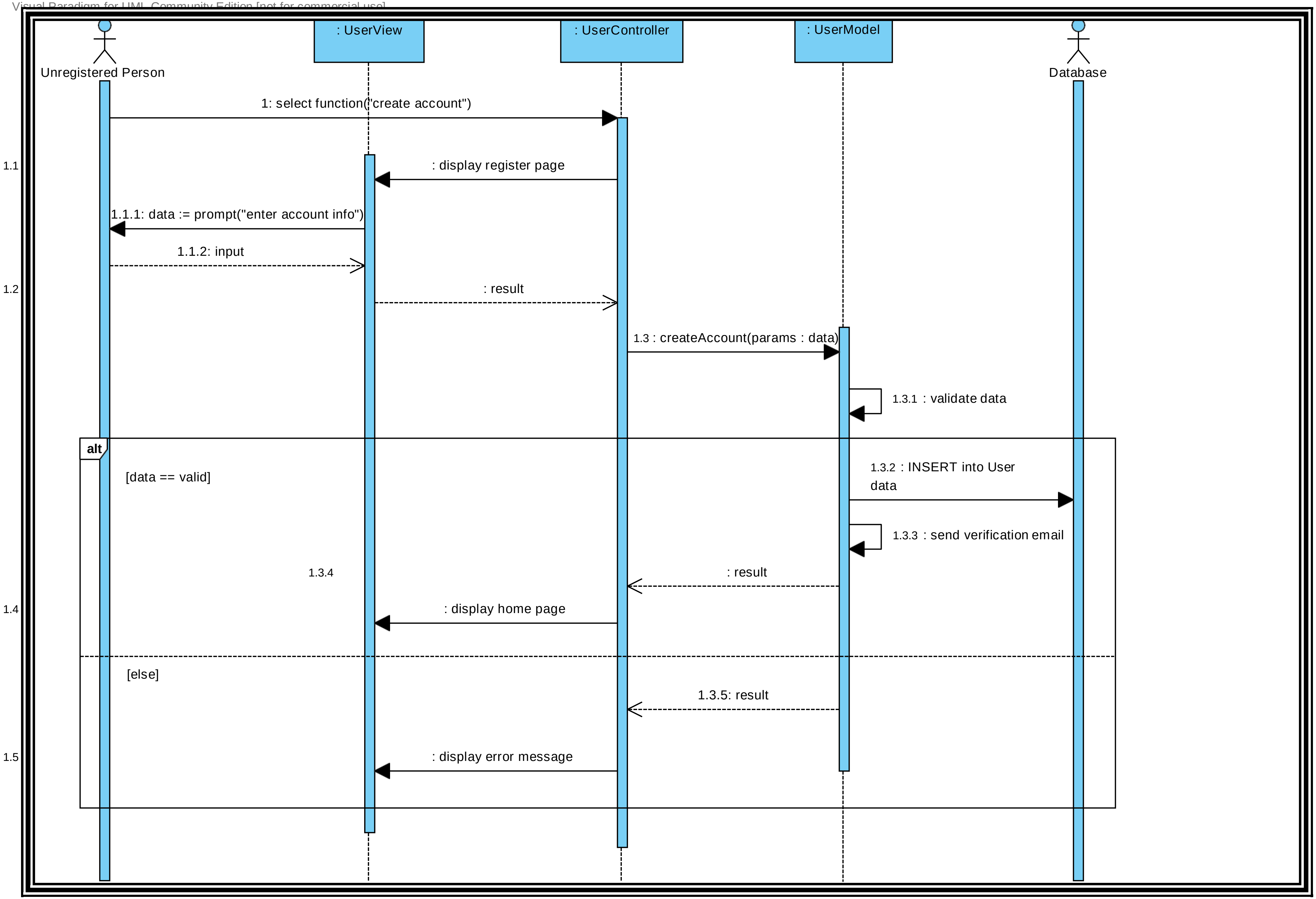
Process: The registered person selects the option to view his or her existing reservations, both past, present, and future. The Controller receives this choice and displays that person's reservations.


Registered Person
: ReservationView
: ReservationController
: ReservationModel
Database
1: select function("View Reservations")
1.1: retrieveReservations
1.1.1: SELECT Reservations WHERE custID
1.1.2: result
1.2: result
1.3: displayReservations
2: reservationID := "edit reservation"
3: editReservation(reservationID)
3.1: goto UC-7: Edit Reservations

UC-5: Register

Goals: To become a registered person and be stored in the database.

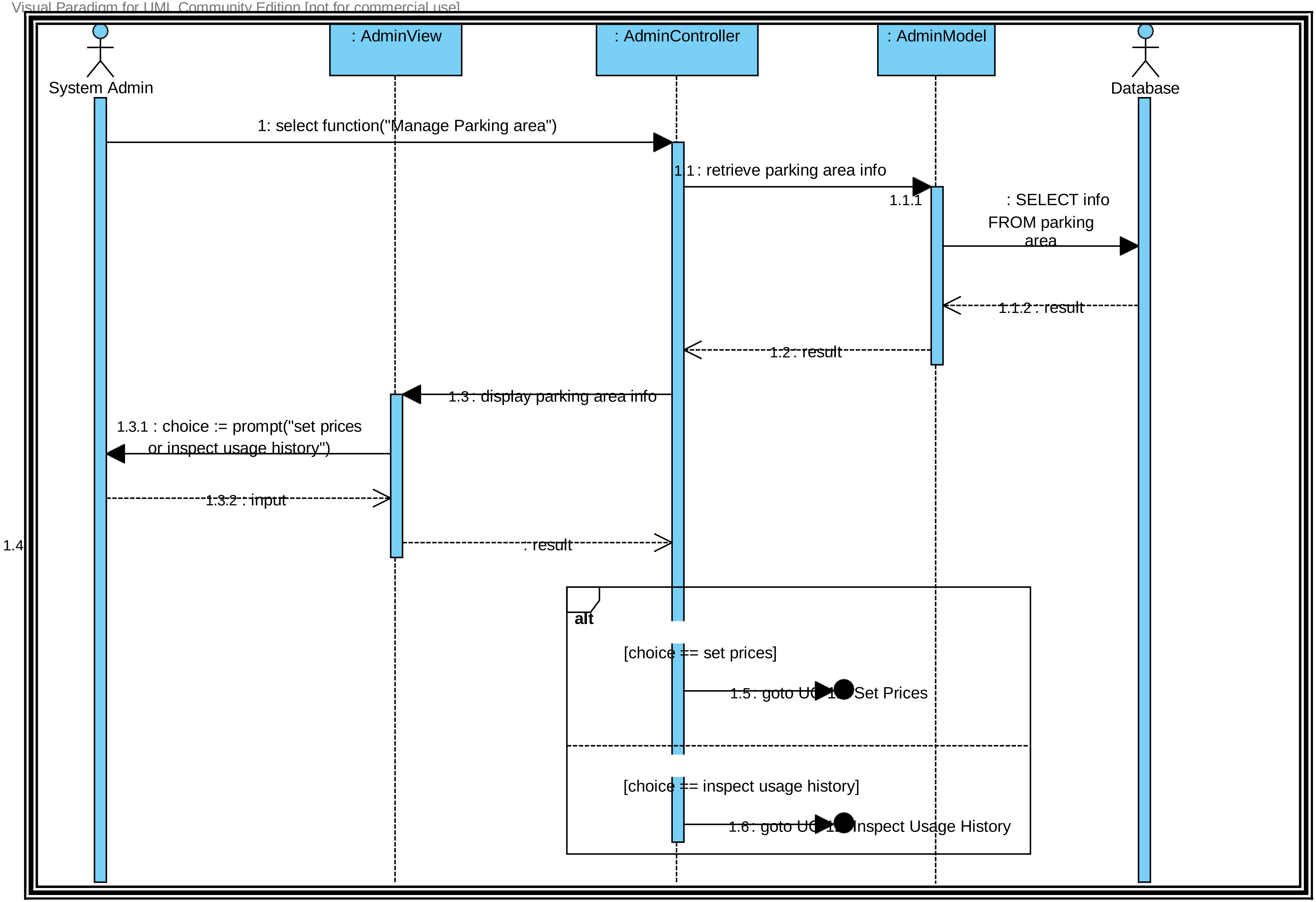
Process: The registered person is prompted to input their account information. The Controller passes the account information to the Model which creates the account in the database. The Controller then calls the View to display a page signifying the success.


Unregistered Person
: UserView
: UserController
: UserModel
Database
1: select function("create account")
1.1
: display register page
1.1.1: data := prompt("enter account info")
1.1.2: input
1.2
: result
1.3 : createAccount(params : data)
1.3.1 : validate data
alt
[data == valid]
1.3.2 : INSERT into User data
1.3.3 : send verification email
1.3.4
: result
1.4
: display home page
[else]
1.3.5: result
1.5
: display error message

UC-6: Manage Parking area

Goals: For a system admin to either set prices or view access history of a parking area (these are both sub-use cases)..

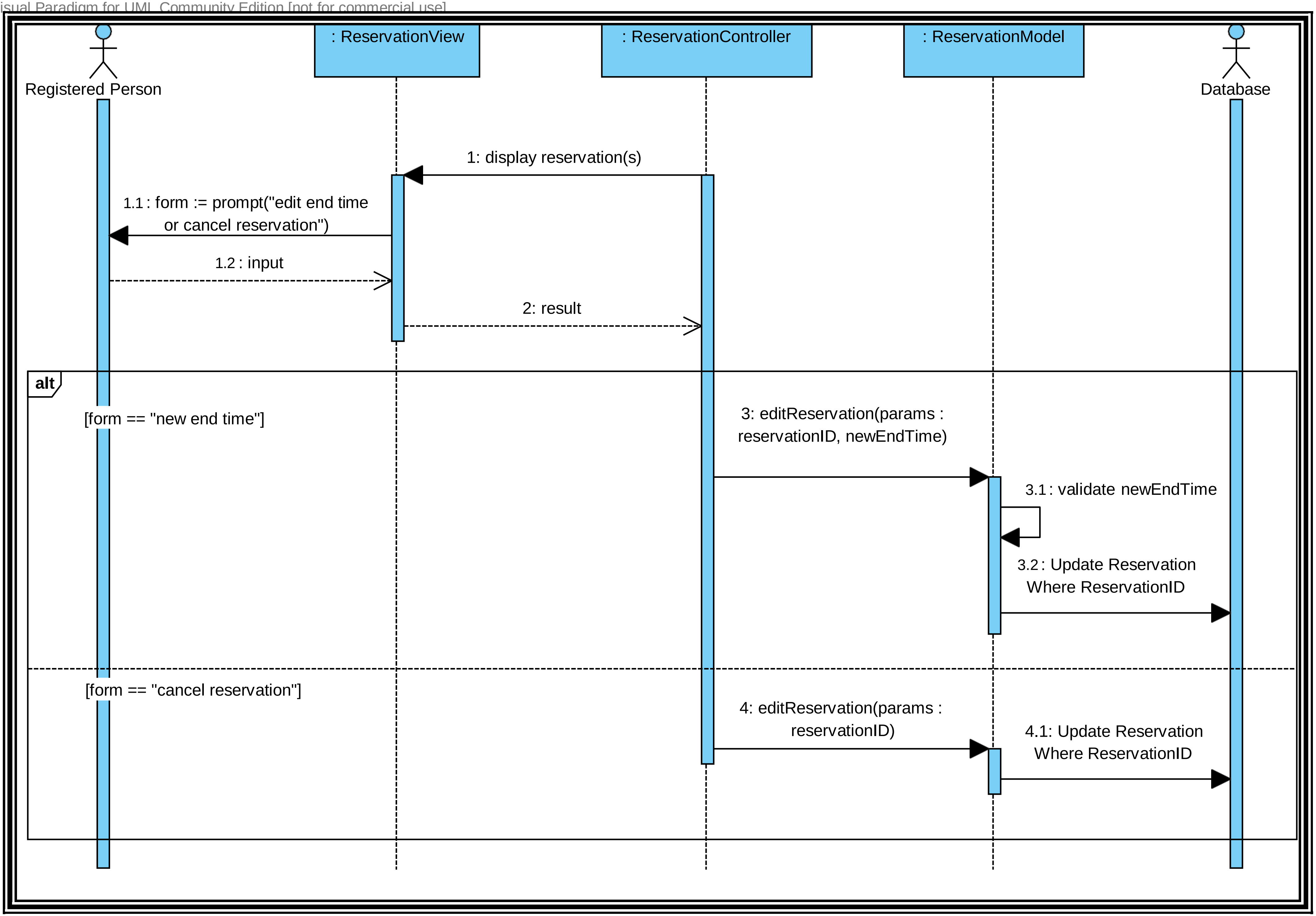
Process: The system admin is prompted to select whether to set prices or view access history for a parking area. The Controller receives the choice and then passes it to to either system interaction diagram 11 or 12.


System Admin
: AdminView
: AdminController
: AdminModel
Database
1: select function("Manage Parking area")
1.1 : retrieve parking area info
1.1.1
: SELECT info
FROM parking
area
1.1.2 : result
1.2 : result
1.3 : display parking area info
1.3.1 : choice := prompt("set prices
or inspect usage history")
1.3.2 : input
1.4
: result
alt
[choice == set prices]
1.5 : goto UC 1
Set Prices
[choice == inspect usage history]
1.6 : goto UC 1
Inspect Usage History

UC-7: Edit Reservation

Goals: To edit or cancel an existing reservation for a registered person.

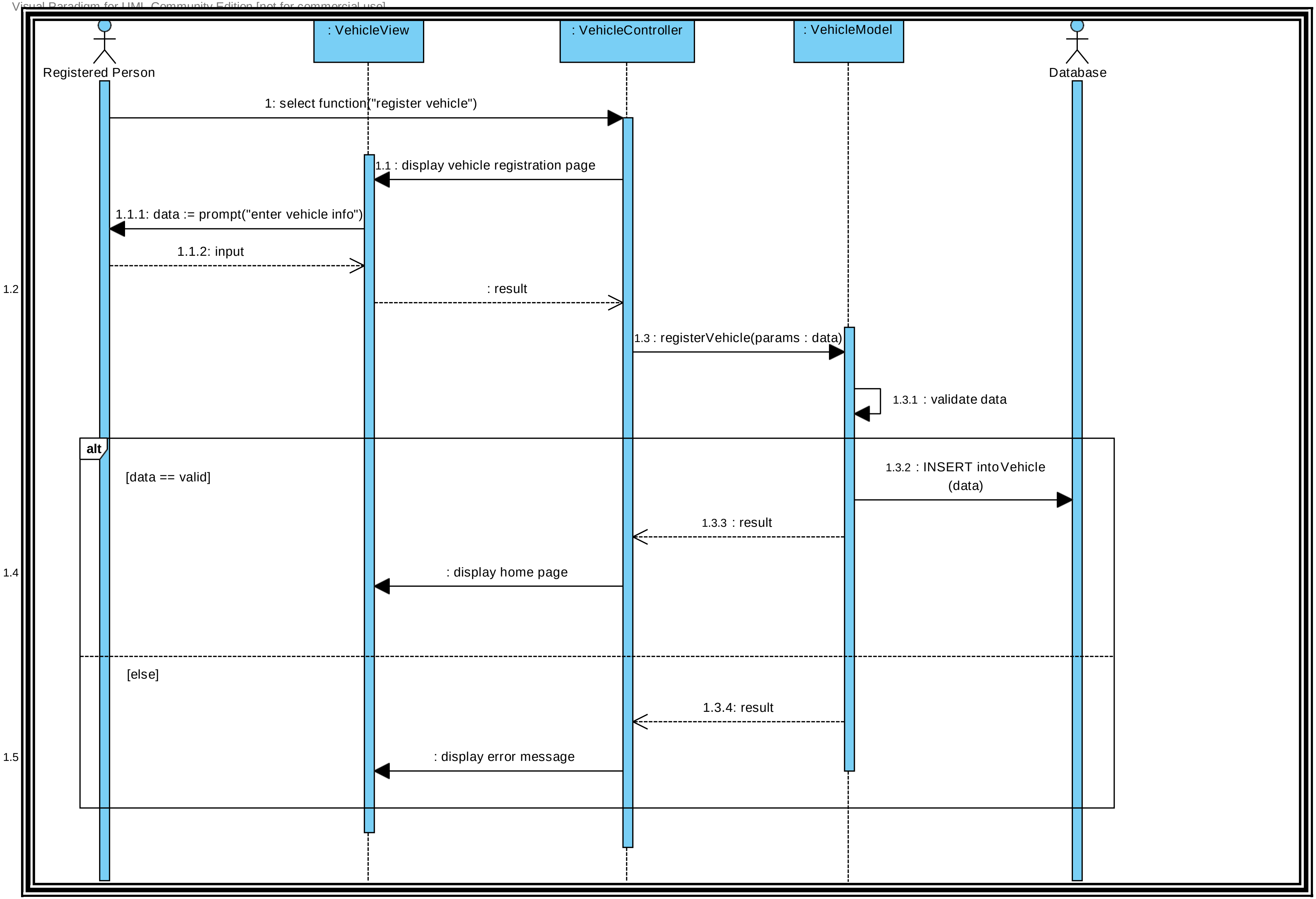
Process: The Registered person either selects an change in the end time of the reservation or selects the cancel reservation option. The Controller receives this information and then passes it to the Model to validate and make the changes in the database. The View displays the successful change.


Registered Person
: ReservationView
: ReservationController
: ReservationModel
Database
1: display reservation(s)
1.1 : form := prompt("edit end time or cancel reservation")
1.2 : input
2: result
alt
[form == "new end time"]
3: editReservation(params : reservationID, newEndTime)
3.1 : validate newEndTime
3.2 : Update Reservation Where ReservationID
[form == "cancel reservation"]
4: editReservation(params : reservationID)
4.1: Update Reservation Where ReservationID

UC-8: Register Vehicle

Goals: To create a vehicle for a Registered Person’s account.

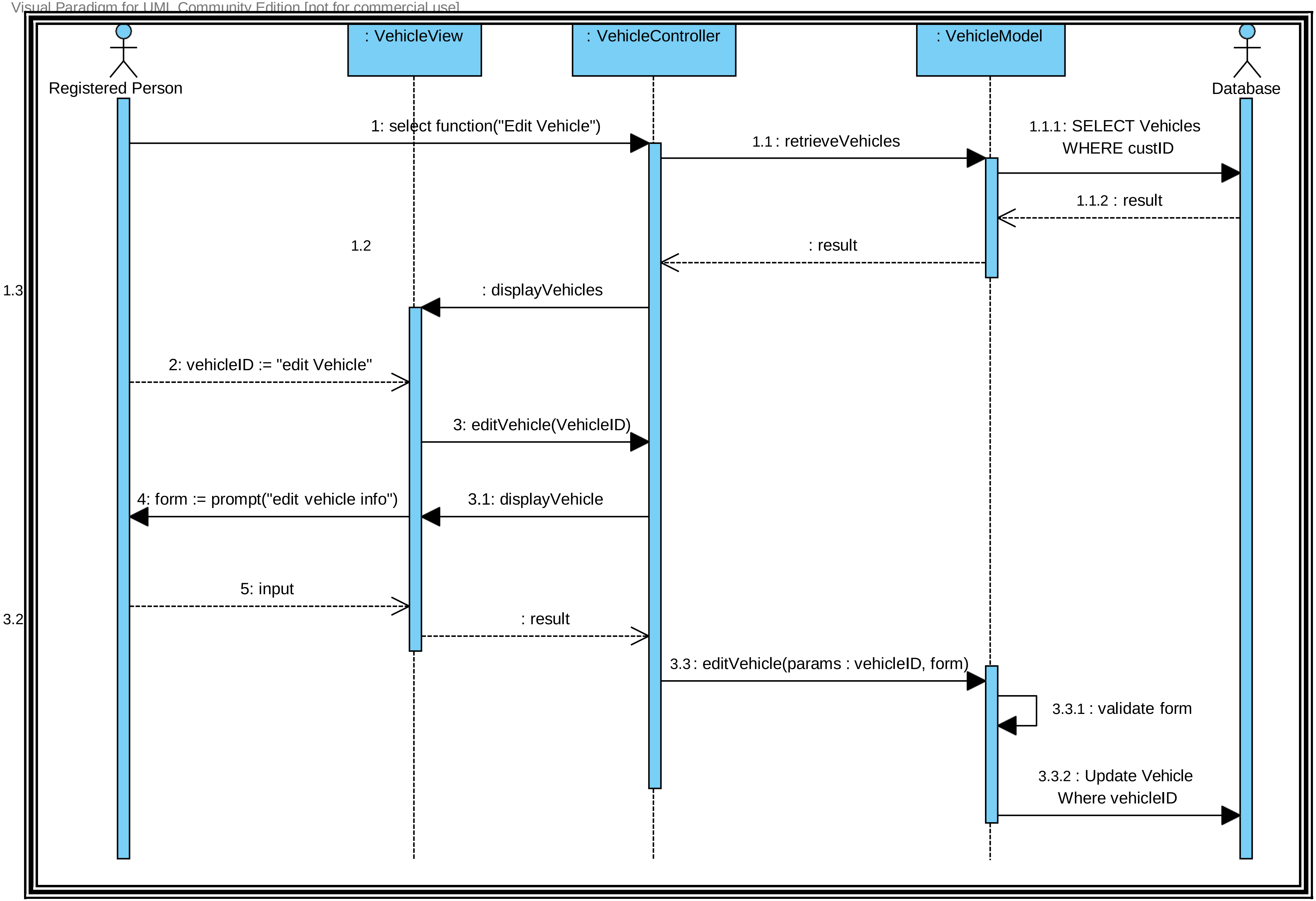
Process: The Registered Person enters the vehicle’s license plate number and state. The Controller receives this information and then passes it to the Model which validates it and then makes the addition to the database. The View displays the successful creation of the vehicle.


Registered Person
: VehicleView
: VehicleController
: VehicleModel
Database
1: select function("register vehicle")
1.1 : display vehicle registration page
1.1.1: data := prompt("enter vehicle info")
1.1.2: input
1.2
: result
1.3 : registerVehicle(params : data)
1.3.1 : validate data
alt
[data == valid]
1.3.2 : INSERT into Vehicle (data)
1.3.3 : result
1.4
: display home page
[else]
1.3.4: result
1.5
: display error message

UC-9: Edit Vehicle

Goals: To edit a registered vehicle's license plate number or state..

Process: The Registered Person enters the vehicle's new license plate number and state. The Controller receives this information and then passes it to the Model which validates it and then makes the addition to the database. The View displays the successful change to the vehicle.


Registered Person
: VehicleView
: VehicleController
: VehicleModel
Database
1: select function("Edit Vehicle")
1.1 : retrieveVehicles
1.1.1: SELECT Vehicles WHERE custID
1.1.2 : result
1.2
: result
1.3
: displayVehicles
2: vehicleID := "edit Vehicle"
3: editVehicle(VehicleID)
4: form := prompt("edit vehicle info")
3.1: displayVehicle
5: input
3.2
: result
3.3 : editVehicle(params : vehicleID, form)
3.3.1 : validate form
3.3.2 : Update Vehicle Where vehicleID

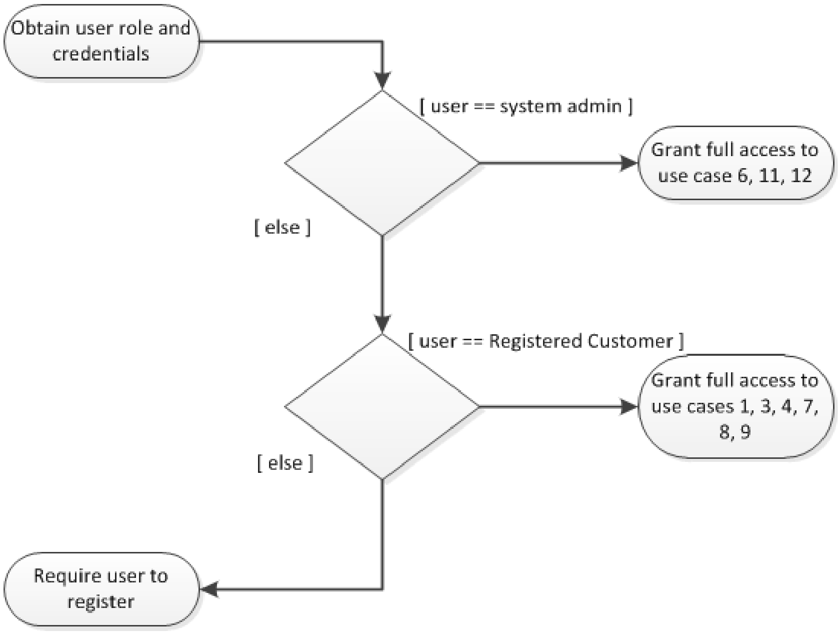
UC-10: Authenticate User

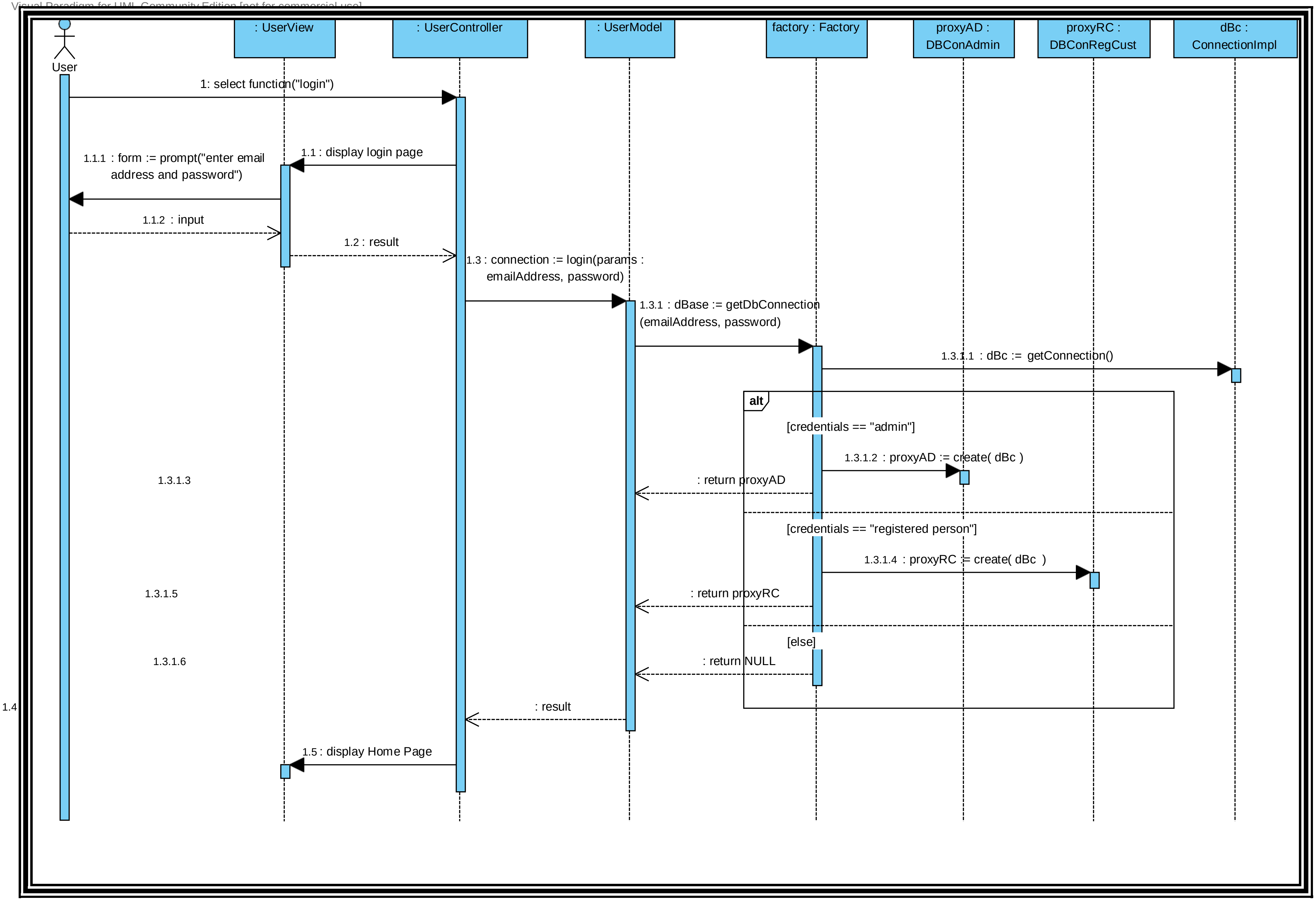
Goals: To determine if a user is registered with the system and has an account in the database.

Process: The user accesses the Interface Page via a web browser, selects log in, and enters their credentials (email and password). The Controller passes the information to the Model which creates a protection proxy based on the user's privileges.

Design Pattern:
We have employed the protection proxy pattern in use case 10, Authenticate User. We decided to use this pattern because different users with different roles log into the system and it would be helpful to assign each user their privileges through a protection proxy. The two roles for the protection proxy are system admin and Registered Person. The diagram below shows which use cases the two different roles have a privilege to participate in.

The advantages to using the protection proxy are that we are able to take the logic for granting privileges away from the models and into a proxy. By doing this, it will be easy in the future to add additional roles and privileges. If we were to keep the system the way it was, then it would be a daunting task to make any changes because all of the complex logic and IF-THEN-ELSE statements involved. Also, the roles and privileges serve as a distraction from the main task of the client and server objects, so adding a protection proxy removes side responsibilities away from the objects and into their own proxy. The protection proxy is also able to protect an object from unauthorized access better than in out previous system because of its simplicity.

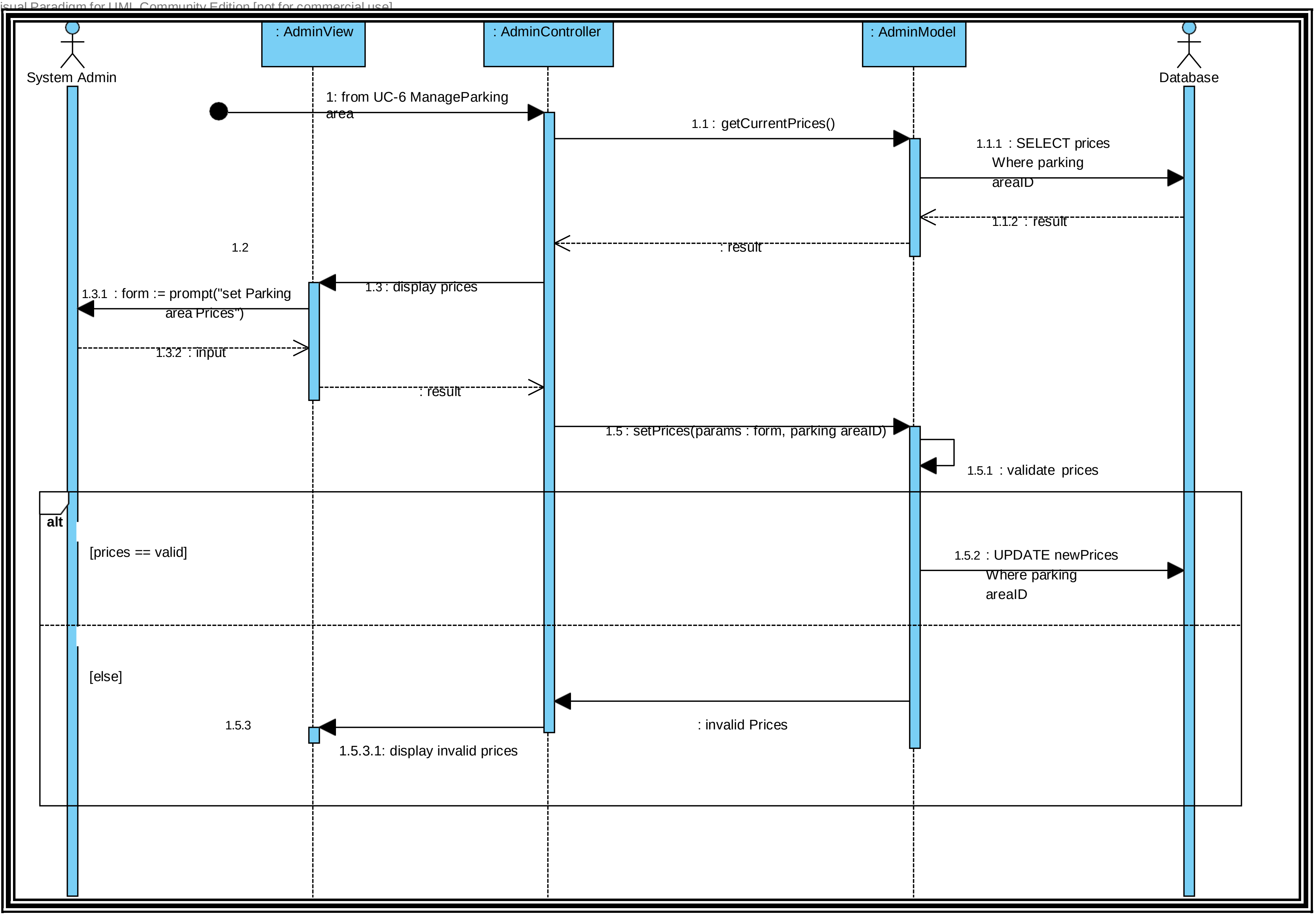

Obtain user role and credentials
[user == system admin]
Grant full access to use case 6, 11, 12
[else]
[user == Registered Customer]
Grant full access to use cases 1, 3, 4, 7, 8, 9
[else]
Require user to register


User
: UserView
: UserController
: UserModel
factory : Factory
proxyAD : DBConAdmin
proxyRC : DBConRegCust
dBc : ConnectionImpl
1: select function("login")
1.1 : display login page
1.1.1 : form := prompt("enter email address and password")
1.1.2 : input
1.2 : result
1.3 : connection := login(params : emailAddress, password)
1.3.1 : dBase := getDbConnection (emailAddress, password)
1.3.1.1 : dBc := getConnection()
alt
[credentials == "admin"]
1.3.1.2 : proxyAD := create(dBc)
1.3.1.3
: return proxyAD
[credentials == "registered person"]
1.3.1.4 : proxyRC := create(dBc)
1.3.1.5
: return proxyRC
[else]
1.3.1.6
: return NULL
1.4
: result
1.5 : display Home Page

UC-11: Set Prices

Goal: To update the prices for parking at the parking area.

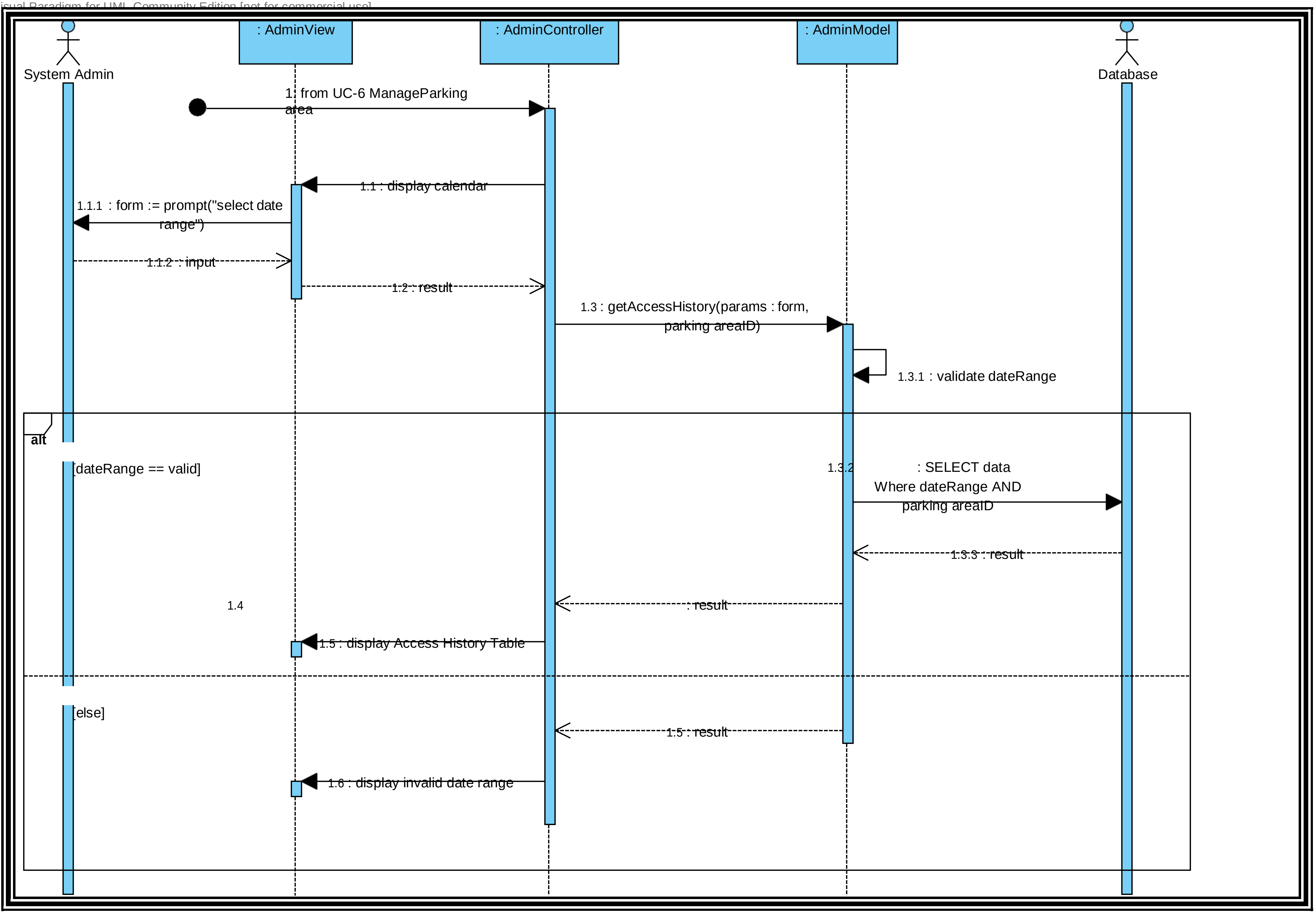
Process: The system admin requests via a web browser to change the prices for a parking area, and is given a web page containing the old pricing information and a form to change to newer prices. Controller takes this form and passes it to the Model, which extracts the new info from it and updates the Database with that pricing data. The web page the system admin sees is then updated with the new prices and a confirmation of the changes.

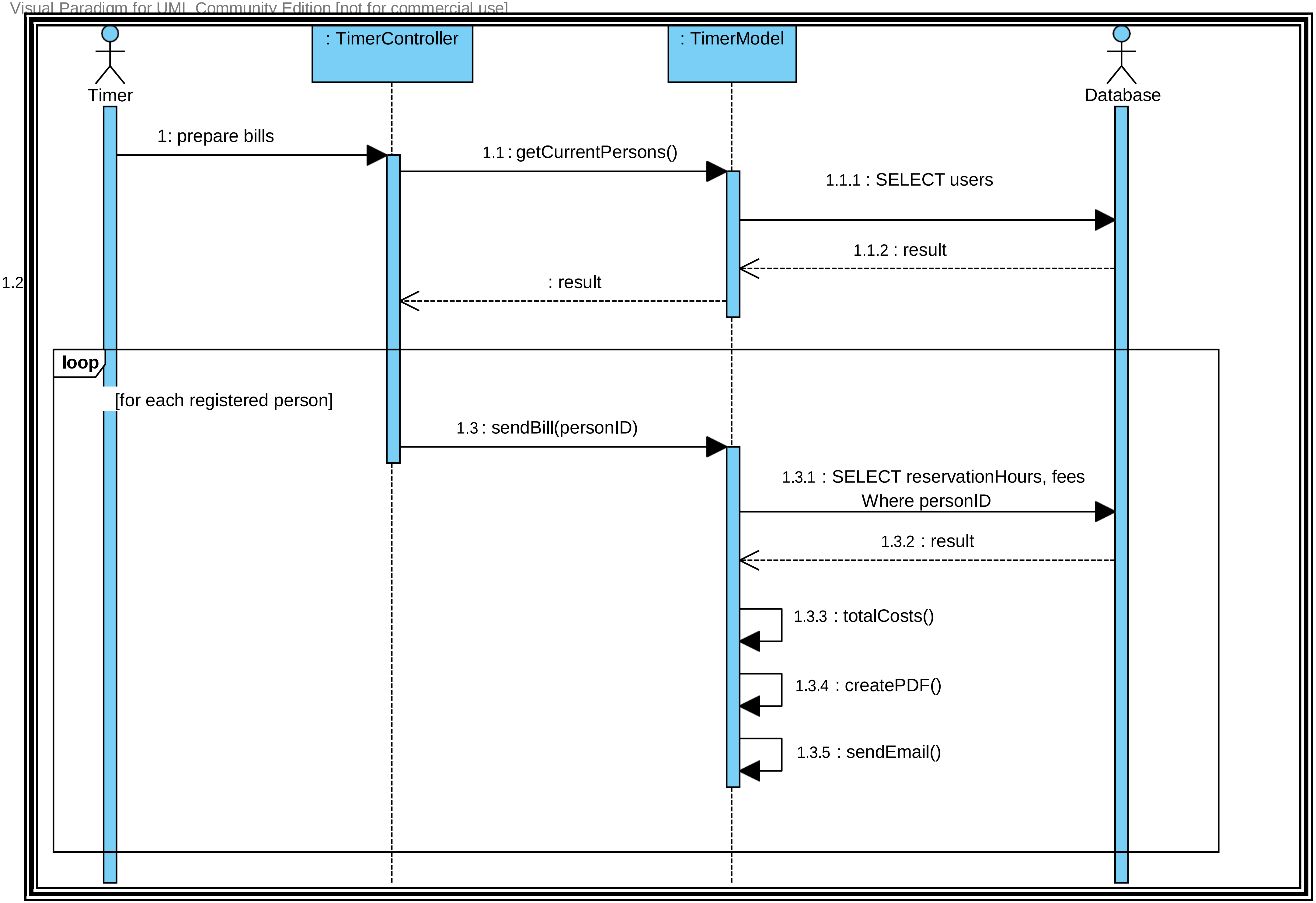

System Admin
: AdminView
: AdminController
: AdminModel
Database
1: from UC-6 ManageParking area
1.1 : getCurrentPrices()
1.1.1 : SELECT prices Where parking areaID
1.1.2 : result
: result
1.2
1.3 : display prices
1.3.1 : form := prompt("set Parking area Prices")
1.3.2 : input
: result
1.4
1.5 : setPrices(params : form, parking areaID)
1.5.1 : validate prices
alt
[prices == valid]
1.5.2 : UPDATE newPrices Where parking areaID
[else]
: invalid Prices
1.5.3
1.5.3.1: display invalid prices

UC-12: Inspect Usage History

Goal: To view the usage history of the parking parking area.

Process: The system admin requests to view the parking parking area usage history, and the Controller requests from Model that the history be pulled from the Database for a certain range of dates input by the system admin. The Model retrieves this data and the View displays the statistics to the system admin for review.


System Admin
: AdminView
: AdminController
: AdminModel
Database
1: from UC-6 ManageParking area
1.1 : display calendar
1.1.1 : form := prompt("select date range")
1.1.2 : input
1.2 : result
1.3 : getAccessHistory(params : form, parking areaID)
1.3.1 : validate dateRange
alt
[dateRange == valid]
1.3.2 : SELECT data Where dateRange AND parking areaID
1.3.3 : result
1.4 : result
1.5 : display Access History Table
[else]
1.5 : result
1.6 : display invalid date range


Timer
: TimerController
: TimerModel
Database
1: prepare bills
1.1 : getCurrentPersons()
1.1.1 : SELECT users
1.1.2 : result
: result
1.2
loop
[for each registered person]
1.3 : sendBill(personID)
1.3.1 : SELECT reservationHours, fees Where personID
1.3.2 : result
1.3.3 : totalCosts()
1.3.4 : createPDF()
1.3.5 : sendEmail()

Class Diagram and Interface Specification

Class Diagrams

For this project, we plan to take advantage of a PHP framework known as Kohana [4]. Therefore, our class diagrams will be a composition between classes that come directly from tables in the Database (see next section, Data Types and Operation Signatures). In addition, we develop a class diagram from our system interaction diagrams. The marriage of these two will be the basis for our system.

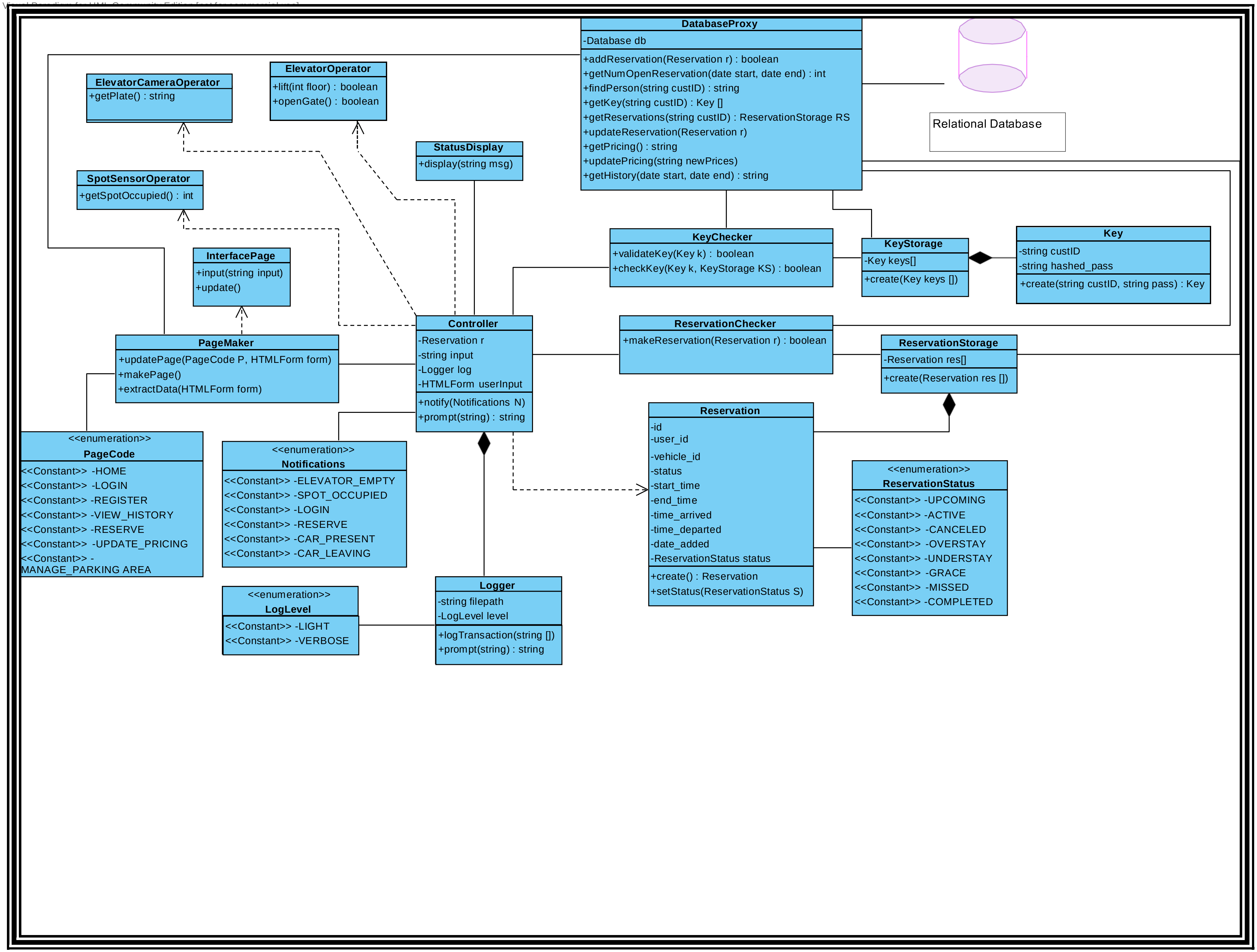
The advantage to using the Kohana framework in this way is that we can now run methods against objects in our program rather than SQL queries against the database. From a programming perspective, this is ideal because it allows to always keep an object-oriented view about our program (treating tables in the database as objects in the program). However, it makes our system interaction diagrams slightly more difficult to translate into a class diagram.

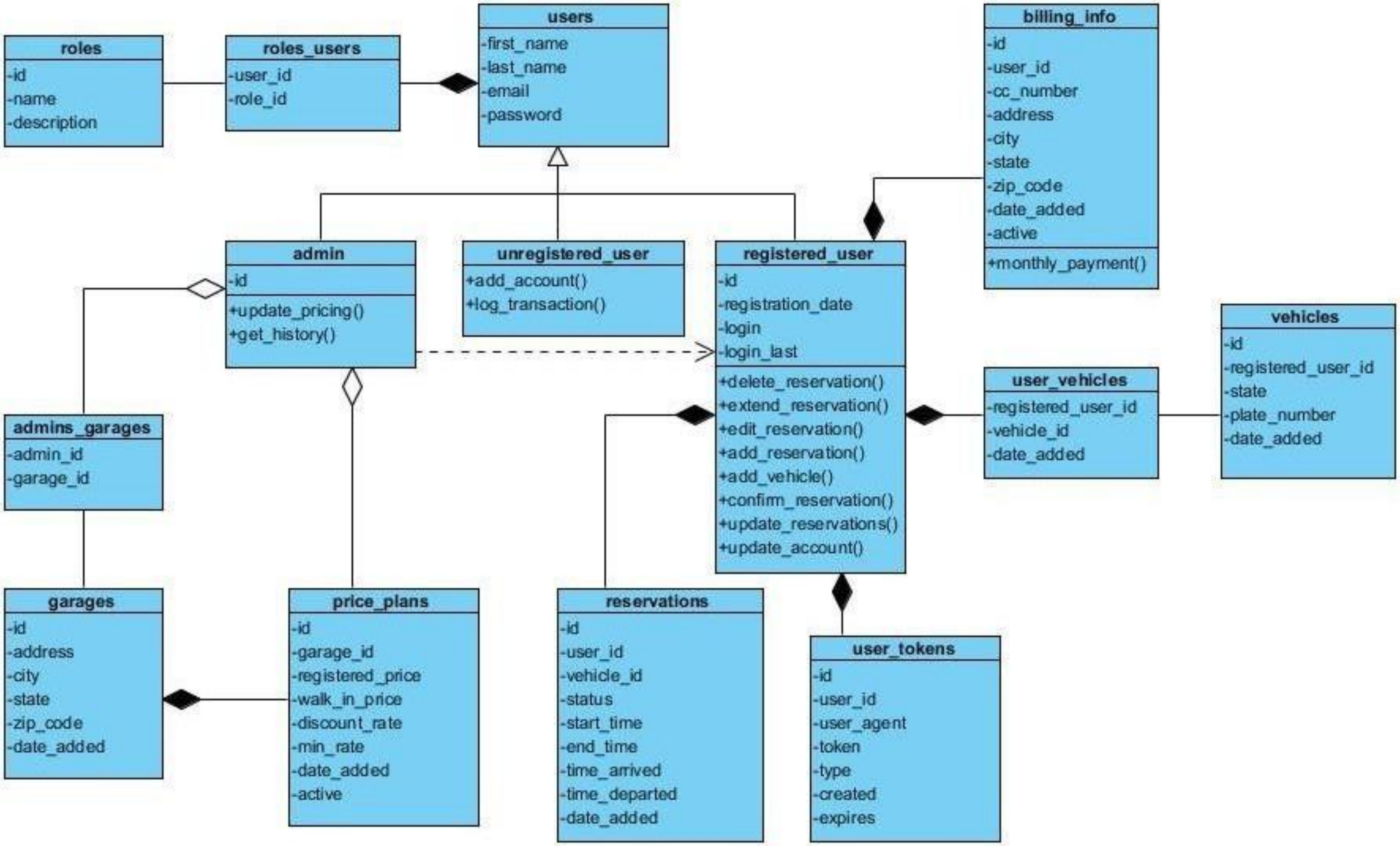
Therefore, we define both a class diagram from our system interaction diagram and also a class diagram for our Kohana framework mapped tables.

To alleviate a lot of the coding needed to implement our system, we will be using an existing, open source PHP framework, Kohana version 3.1. The Kohana framework works on the MVC (model, view, controller) architecture. The basic idea of this system architecture is that business logic is stored in models, person-facing presentation is coded in views, and controller's handle the interaction between the views (persons) and the models (our system).

Within Kohana, there is exists an ORM (object relational mapping) library that abstracts a lot of the database queries as objects. Within the ORM there are a lot of common methods, such as `save()`, `create()`, and `edit()`. These stock methods will make it easy to implement our system.

Below is a sample of the PHP code inside the Kohana Framework that we might use in our design. It consists of an example model for reservations in the parking parking area. This model directly maps to the reservation table in our database, and is abstracted using the ORM.


DatabaseProxy
-Database db
+addReservation(Reservation r) : boolean
+getNumOpenReservation(date start, date end) : int
+findPerson(string custID) : string
+getKey(string custID) : Key []
+getReservations(string custID) : ReservationStorage RS
+updateReservation(Reservation r)
+getPricing() : string
+updatePricing(string newPrices)
+getHistory(date start, date end) : string
Relational Database
ElevatorCameraOperator
+getPlate() : string
ElevatorOperator
+lift(int floor) : boolean
+openGate() : boolean
StatusDisplay
+display(string msg)
SpotSensorOperator
+getSpotOccupied() : int
InterfacePage
+input(string input)
+update()
KeyChecker
+validateKey(Key k) : boolean
+checkKey(Key k, KeyStorage KS) : boolean
KeyStorage
-Key keys[]
+create(Key keys [])
Key
-string custID
-string hashed_pass
+create(string custID, string pass) : Key
Controller
-Reservation r
-string input
-Logger log
-HTMLForm userInput
+notify(Notifications N)
+prompt(string) : string
ReservationChecker
+makeReservation(Reservation r) : boolean
ReservationStorage
-Reservation res[]
+create(Reservation res [])
PageMaker
+updatePage(PageCode P, HTMLForm form)
+makePage()
+extractData(HTMLForm form)
<<enumeration>>
PageCode
<<Constant>> -HOME
<<Constant>> -LOGIN
<<Constant>> -REGISTER
<<Constant>> -VIEW_HISTORY
<<Constant>> -RESERVE
<<Constant>> -UPDATE_PRICING
<<Constant>> -MANAGE_PARKING AREA
<<enumeration>>
Notifications
<<Constant>> -ELEVATOR_EMPTY
<<Constant>> -SPOT_OCCUPIED
<<Constant>> -LOGIN
<<Constant>> -RESERVE
<<Constant>> -CAR_PRESENT
<<Constant>> -CAR_LEAVING
Reservation
-id
-user_id
-vehicle_id
-status
-start_time
-end_time
-time_arrived
-time_departed
-date_added
-ReservationStatus status
+create() : Reservation
+setStatus(ReservationStatus S)
<<enumeration>>
ReservationStatus
<<Constant>> -UPCOMING
<<Constant>> -ACTIVE
<<Constant>> -CANCELED
<<Constant>> -OVERSTAY
<<Constant>> -UNDERSTAY
<<Constant>> -GRACE
<<Constant>> -MISSED
<<Constant>> -COMPLETED
<<enumeration>>
LogLevel
<<Constant>> -LIGHT
<<Constant>> -VERBOSE
Logger
-string filepath
-LogLevel level
+logTransaction(string [])
+prompt(string) : string


roles
-id
-name
-description
roles_users
-user_id
-role_id
users
-first_name
-last_name
-email
-password
billing_info
-id
-user_id
-cc_number
-address
-city
-state
-zip_code
-date_added
-active
+monthly_payment()
admin
-id
+update_pricing()
+get_history()
unregistered_user
+add_account()
+log_transaction()
registered_user
-id
-registration_date
-login
-login_last
+delete_reservation()
+extend_reservation()
+edit_reservation()
+add_reservation()
+add_vehicle()
+confirm_reservation()
+update_reservations()
+update_account()
vehicles
-id
-registered_user_id
-state
-plate_number
-date_added
user_vehicles
-registered_user_id
-vehicle_id
-date_added
admins_garages
-admin_id
-garage_id
garages
-id
-address
-city
-state
-zip_code
-date_added
price_plans
-id
-garage_id
-registered_price
-walk_in_price
-discount_rate
-min_rate
-date_added
-active
reservations
-id
-user_id
-vehicle_id
-status
-start_time
-end_time
-time_arrived
-time_departed
-date_added
user_tokens
-id
-user_id
-user_agent
-token
-type
-created
-expires

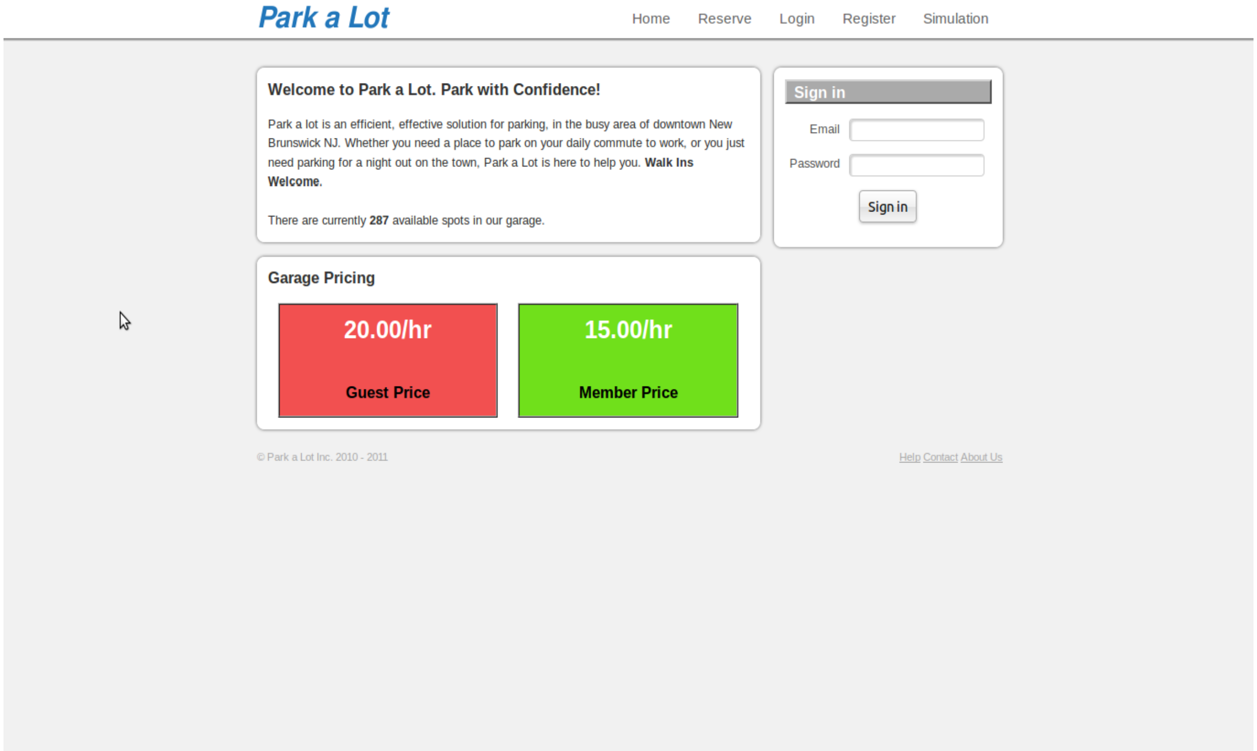
User Interface Design and Implementation

A few significant additions/edits were made to the user interface to increase the user experience. All mock-ups can be seen live at http://www.park-a-lot.vacau.com/. The design is meant to reduce user effort to a minimum, by providing a sleek and reduced graphical interface that is simple to understand (no screen clutter or extraneous information).

The differences between our previous design and our current design go as follows:

Home Page

Since we did not actually implement the multi-parking area feature, we've removed the Google Map from our homepage. We've also added a price breakdown on our home page so persons can easily see our current prices for members and guests alike.



Welcome to Park a Lot. Park with Confidence!

Park a lot is an efficient, effective solution for parking, in the busy area of downtown New Brunswick NJ. Whether you need a place to park on your daily commute to work, or you just need parking for a night out on the town, Park a Lot is here to help you. **Walk Ins Welcome.**

There are currently **287** available spots in our garage.

Garage Pricing

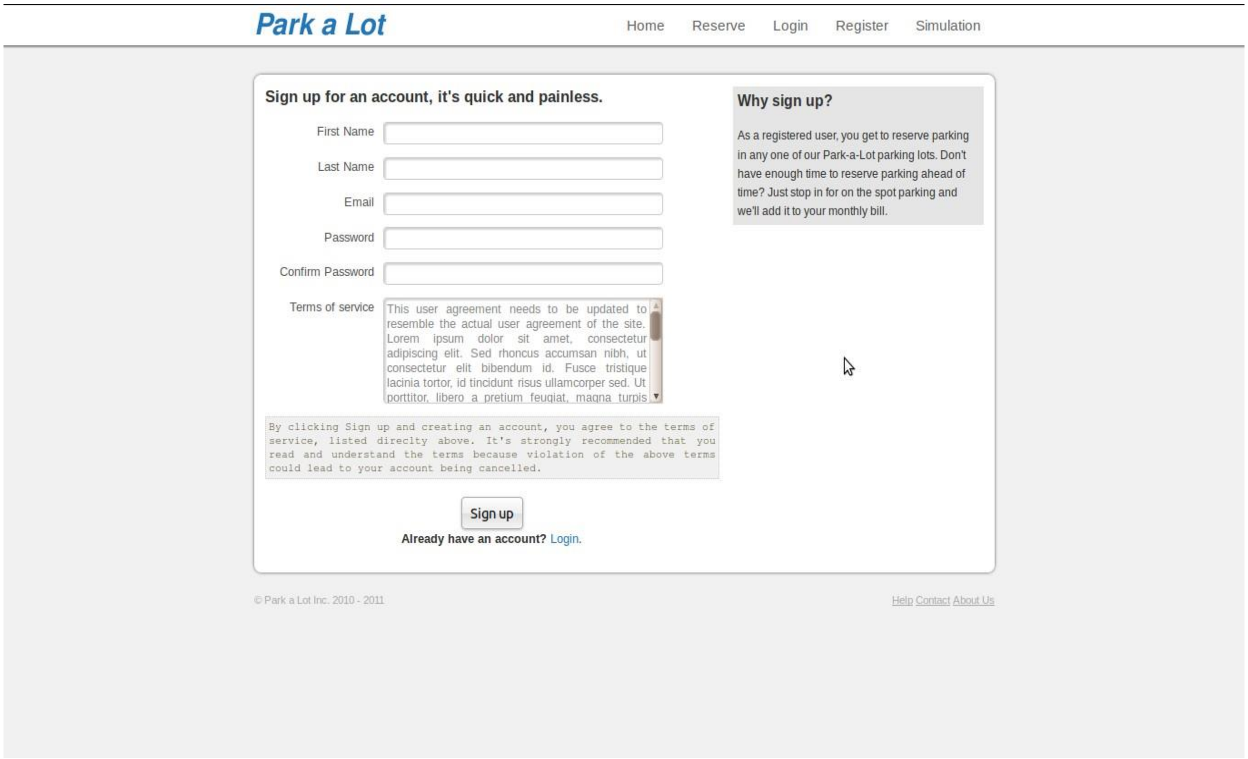
20.00/hr Guest Price	15.00/hr Member Price

© Park a Lot Inc. 2010 - 2011

Help Contact About Us

User Registration

We've eased up on our user registration form, making it easier and quicker for users to sign up. We no longer require any credit card information at sign up.



Sign up for an account, it's quick and painless.

First Name

Last Name

Email

Password

Confirm Password

Terms of service

This user agreement needs to be updated to resemble the actual user agreement of the site. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed rhoncus accumsan nibh, ut consectetur elit bibendum id. Fusce tristique lacinia tortor, id tincidunt risus ullamcorper sed. Ut porttitor, libero a pretium feugiat, magna turpis

```
By clicking Sign up and creating an account, you agree to the terms of
service, listed direclty above. It's strongly recommended that you
read and understand the terms because violation of the above terms
could lead to your account being cancelled.
```

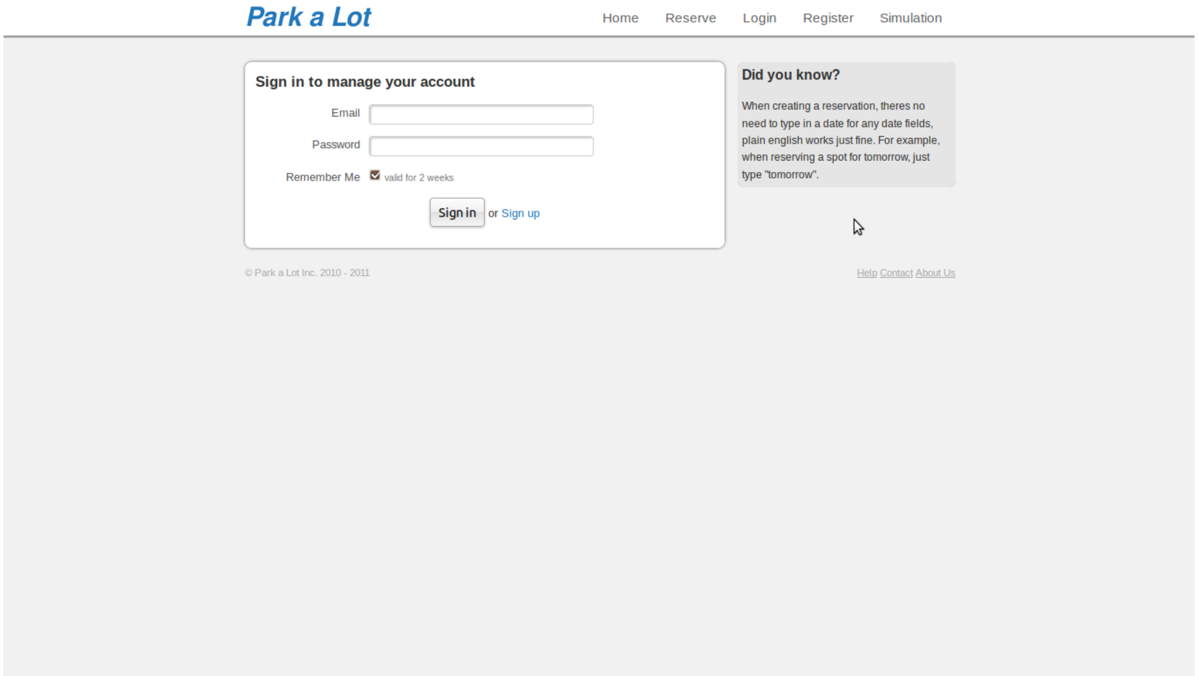
Already have an account? Login.

Why sign up?

As a registered user, you get to reserve parking in any one of our Park-a-Lot parking lots. Don't have enough time to reserve parking ahead of time? Just stop in for on the spot parking and we'll add it to your monthly bill.

User Login

We did not change much with user login, except added a few site-wide tips to our login screen. The login screen as seen from our home page remains the same, but we added a few helpful tips and FAQ to our main login screen.



Sign in to manage your account

Email

Password

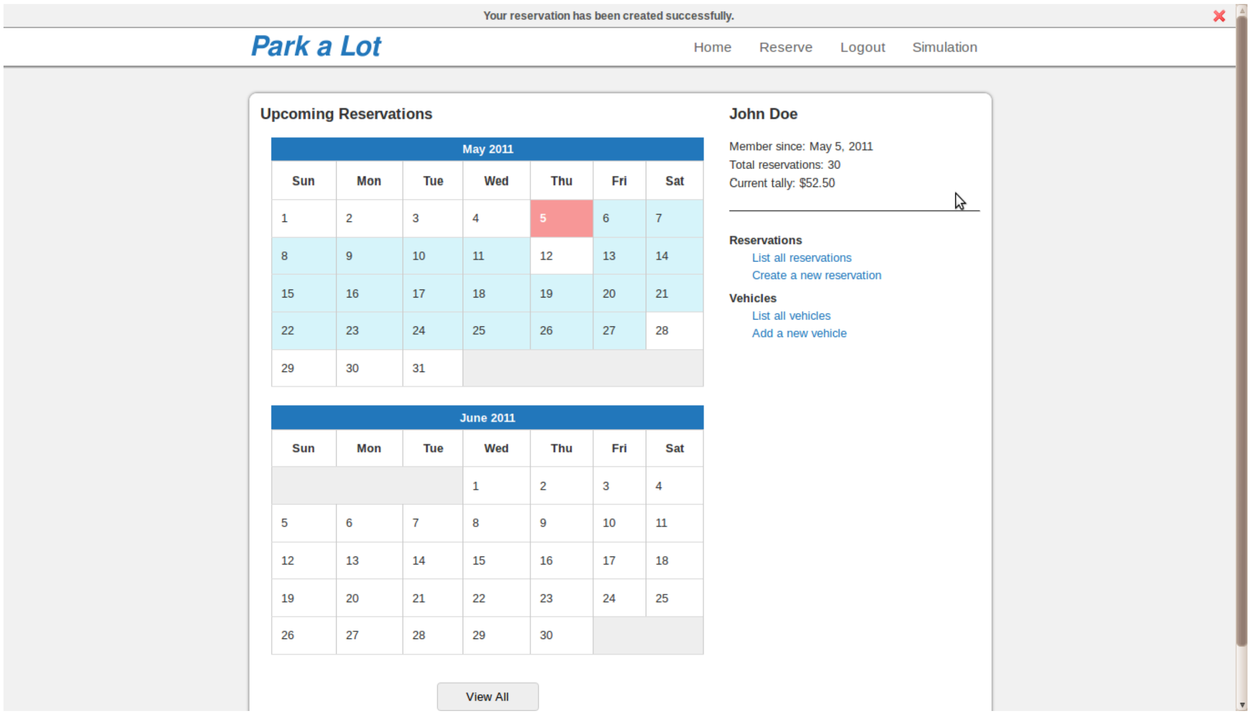
Remember Me ☑ valid for 2 weeks

Did you know?

When creating a reservation, theres no need to type in a date for any date fields, plain english works just fine. For example, when reserving a spot for tomorrow, just type "tomorrow".

User Profile

The user profile page contains links to all important pages from a users standpoint. From there, they can go to the create reservation page, create vehicle page, list reservations page, list vehicles page, as well as view any reservations for a specific day by clicking on a date from the calendars shown, which summarize which days the user has a reservation in the next 2 months. Also, from the home page, the user can see how much they have on their current monthly bill to date. Every action that the user takes, whether it be add a new reservation, cancel an existing one, etc. they are redirected back to their profile and a popup notification is shown at the very top of the page to provide feedback.



Your reservation has been created successfully.

Upcoming Reservations

May 2011

Sun	Mon	Tue	Wed	Thu	Fri	Sat
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

June 2011

Sun	Mon	Tue	Wed	Thu	Fri	Sat
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

John Doe

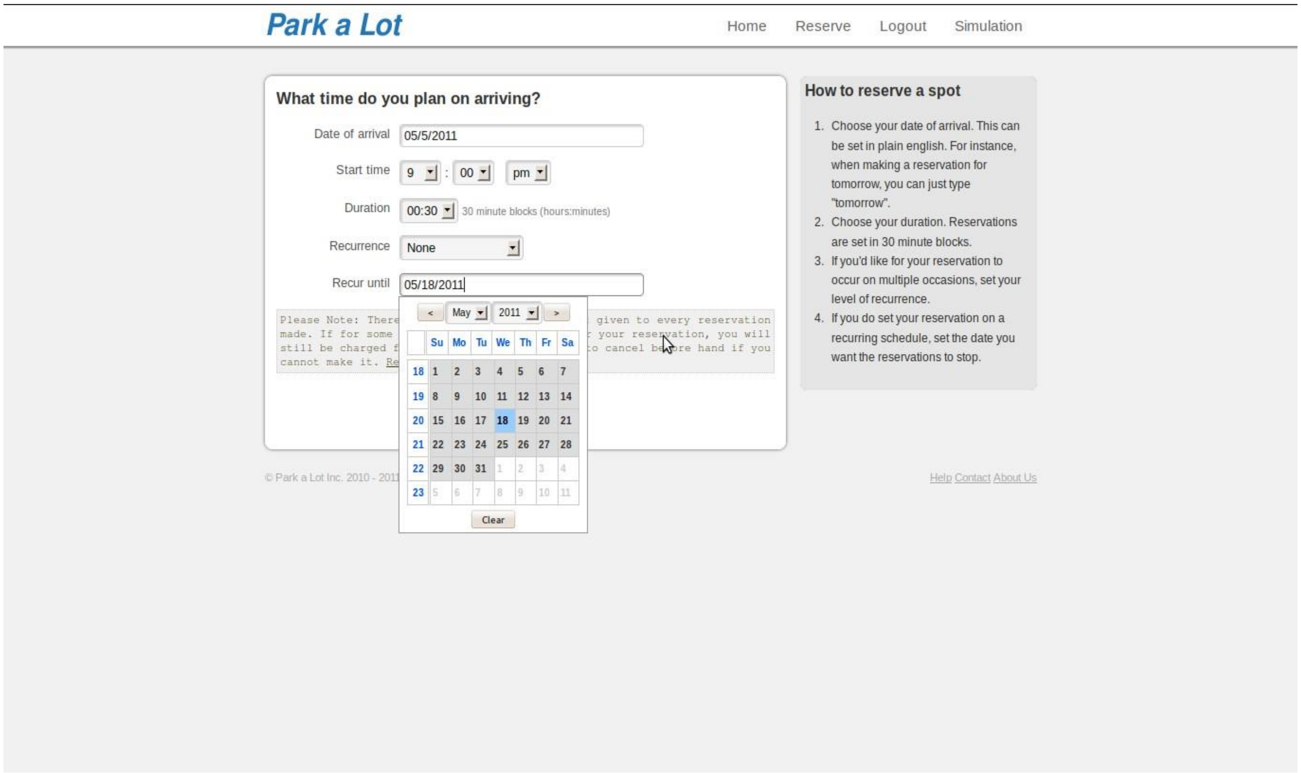
Member since: May 5, 2011
Total reservations: 30
Current tally: $52.50

Reservations
- List all reservations
- Create a new reservation

Vehicles
- List all vehicles
- Add a new vehicle

Create Reservation

The create reservation page was updated to add popup calendars to each field that takes a date input to make it easier for the user.



What time do you plan on arriving?

Date of arrival 05/5/2011

Start time 9 : 00 pm

Duration 00:30 30 minute blocks (hours:minutes)

Recurrence None

Recur until 05/18/2011

```
Please Note: There          given to every reservation
made. If for some          r your reservation, you will
still be charged f          to cancel before hand if you
cannot make it. Re
```

How to reserve a spot

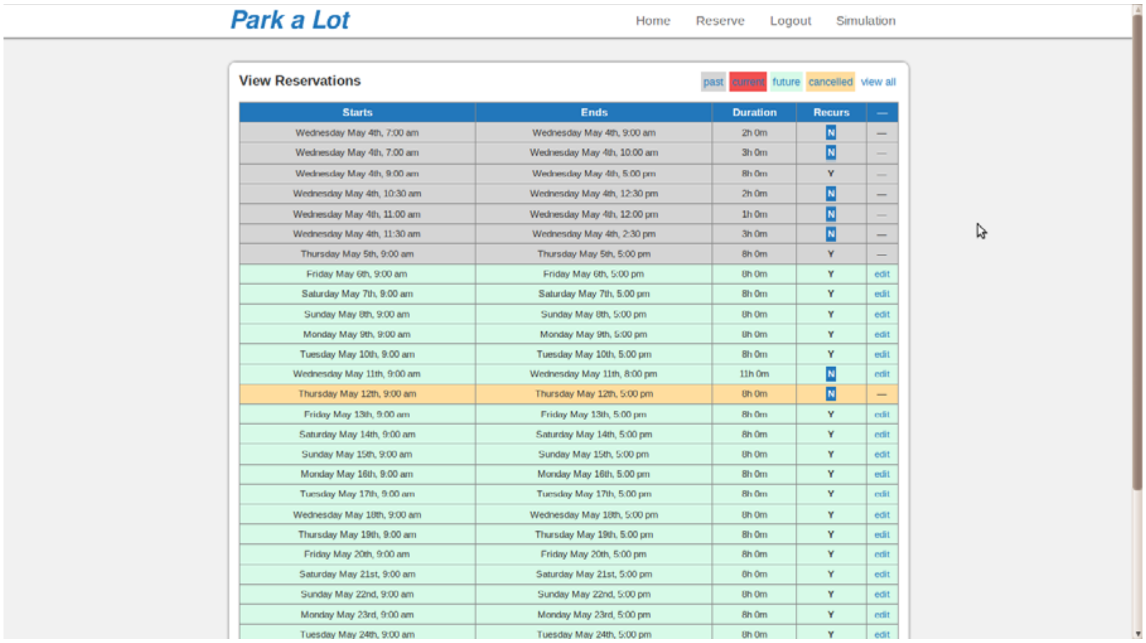
1. Choose your date of arrival. This can be set in plain english. For instance, when making a reservation for tomorrow, you can just type "tomorrow".
2. Choose your duration. Reservations are set in 30 minute blocks.
3. If you'd like for your reservation to occur on multiple occasions, set your level of recurrence.
4. If you do set your reservation on a recurring schedule, set the date you want the reservations to stop.

© Park a Lot Inc. 2010 - 2011

Help Contact About Us

List Reservations

The list reservations page lists in more detail all of the users reservations, past, present, and future. From here, they can choose to edit/cancel any reservation which can still be edited/updated. each reservation is color coded to easily distinguish between past/present/future/cancelled reservations.

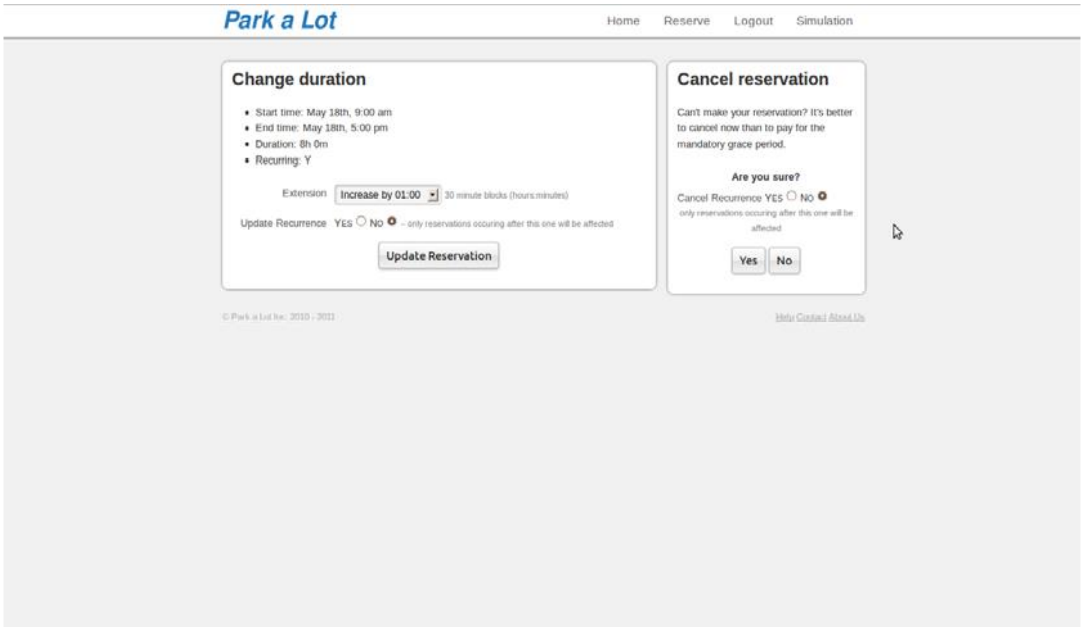


View Reservations

Starts	Ends	Duration	Recurs	—
Wednesday May 4th, 7:00 am	Wednesday May 4th, 9:00 am	2h 0m	N	—
Wednesday May 4th, 7:00 am	Wednesday May 4th, 10:00 am	3h 0m	N	—
Wednesday May 4th, 9:00 am	Wednesday May 4th, 5:00 pm	8h 0m	Y	—
Wednesday May 4th, 10:30 am	Wednesday May 4th, 12:30 pm	2h 0m	N	—
Wednesday May 4th, 11:00 am	Wednesday May 4th, 12:00 pm	1h 0m	N	—
Wednesday May 4th, 11:30 am	Wednesday May 4th, 2:30 pm	3h 0m	N	—
Thursday May 5th, 9:00 am	Thursday May 5th, 5:00 pm	8h 0m	Y	—
Friday May 6th, 9:00 am	Friday May 6th, 5:00 pm	8h 0m	Y	edit
Saturday May 7th, 9:00 am	Saturday May 7th, 5:00 pm	8h 0m	Y	edit
Sunday May 8th, 9:00 am	Sunday May 8th, 5:00 pm	8h 0m	Y	edit
Monday May 9th, 9:00 am	Monday May 9th, 5:00 pm	8h 0m	Y	edit
Tuesday May 10th, 9:00 am	Tuesday May 10th, 5:00 pm	8h 0m	Y	edit
Wednesday May 11th, 9:00 am	Wednesday May 11th, 8:00 pm	11h 0m	N	edit
Thursday May 12th, 9:00 am	Thursday May 12th, 5:00 pm	8h 0m	N	—
Friday May 13th, 9:00 am	Friday May 13th, 5:00 pm	8h 0m	Y	edit
Saturday May 14th, 9:00 am	Saturday May 14th, 5:00 pm	8h 0m	Y	edit
Sunday May 15th, 9:00 am	Sunday May 15th, 5:00 pm	8h 0m	Y	edit
Monday May 16th, 9:00 am	Monday May 16th, 5:00 pm	8h 0m	Y	edit
Tuesday May 17th, 9:00 am	Tuesday May 17th, 5:00 pm	8h 0m	Y	edit
Wednesday May 18th, 9:00 am	Wednesday May 18th, 5:00 pm	8h 0m	Y	edit
Thursday May 19th, 9:00 am	Thursday May 19th, 5:00 pm	8h 0m	Y	edit
Friday May 20th, 9:00 am	Friday May 20th, 5:00 pm	8h 0m	Y	edit
Saturday May 21st, 9:00 am	Saturday May 21st, 5:00 pm	8h 0m	Y	edit
Sunday May 22nd, 9:00 am	Sunday May 22nd, 5:00 pm	8h 0m	Y	edit
Monday May 23rd, 9:00 am	Monday May 23rd, 5:00 pm	8h 0m	Y	edit
Tuesday May 24th, 9:00 am	Tuesday May 24th, 5:00 pm	8h 0m	Y	edit

Edit/Cancel Reservation

From this page, users can edit or cancel a reservation. Reservations can only be cancelled if done so at least 30 minutes prior to the beginning of the reservation, where as reservations can be edited if done so at least 30 minutes before they end. If a user comes to this page after the allowable cancellation period, the cancel action will not be shown. We also added options to edit/cancel all reservations following the one being edited/cancelled if the one being edited/cancelled is a recurring reservation.



Change duration

- Start time: May 18th, 9:00 am
- End time: May 18th, 5:00 pm
- Duration: 8h 0m
- Recurring: Y

Extension Increase by 01:00 30 minute blocks (hours:minutes)

Update Recurrence YES ○ NO ● – only reservations occuring after this one will be affected

Cancel reservation

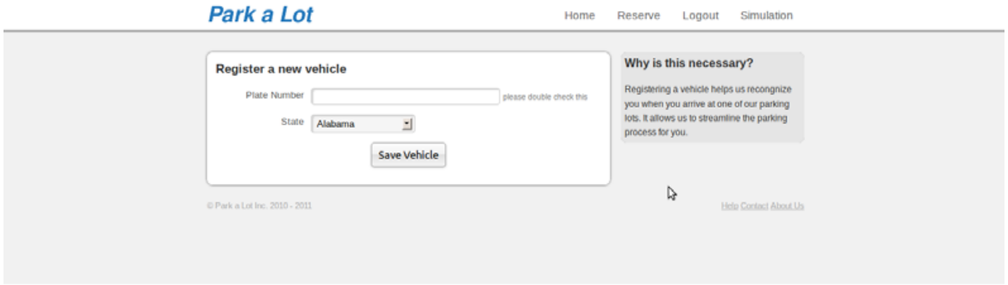
Can't make your reservation? It's better to cancel now than to pay for the mandatory grace period.

Are you sure?

Cancel Recurrence YES ○ NO ●
only reservations occuring after this one will be affected

Add Vehicle

The add vehicle page is where the user goes to register a new vehicle with their account. These vehicles are recognized when the user arrives for quick and easy entrance into the parking area.



Register a new vehicle

Plate Number please double check this

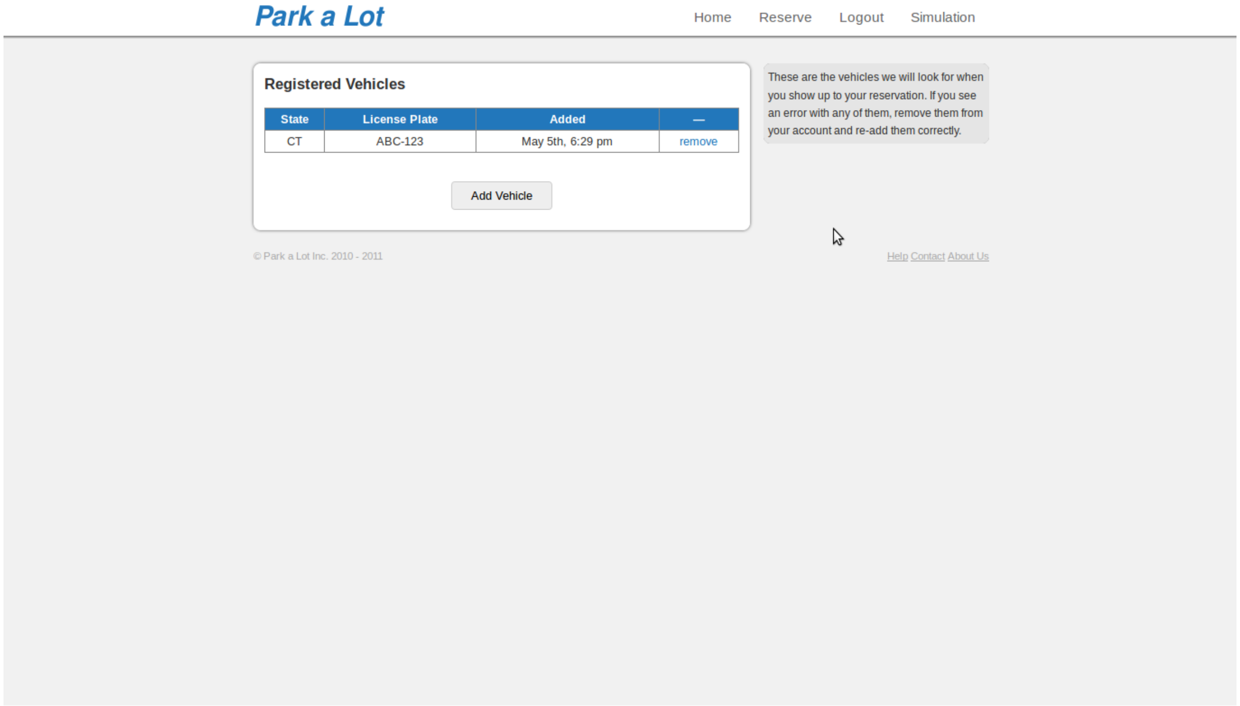
State Alabama

Why is this necessary?

Registering a vehicle helps us recongnize you when you arrive at one of our parking lots. It allows us to streamline the parking process for you.

List Vehicles

The list vehicles lists all vehicles currently registered with the users account.



Registered Vehicles

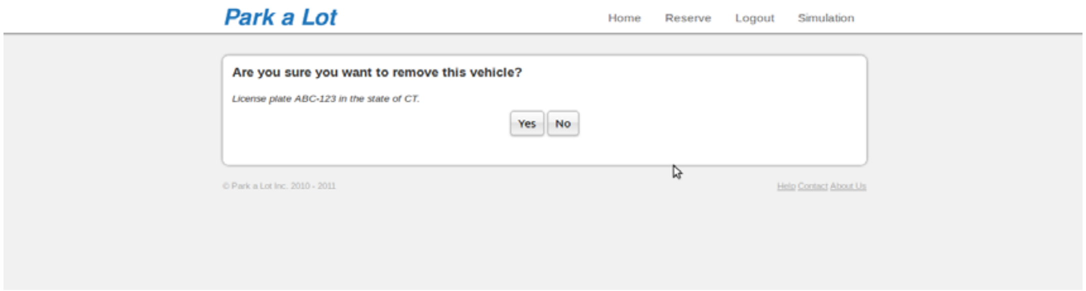
State	License Plate	Added	—
CT	ABC-123	May 5th, 6:29 pm	remove

These are the vehicles we will look for when you show up to your reservation. If you see an error with any of them, remove them from your account and re-add them correctly.

© Park a Lot Inc. 2010 - 2011

Remove Vehicle

From this page, users can de-register any of their vehicles. This page just asks for confirmation, as well as shows information about the vehicle being removed.



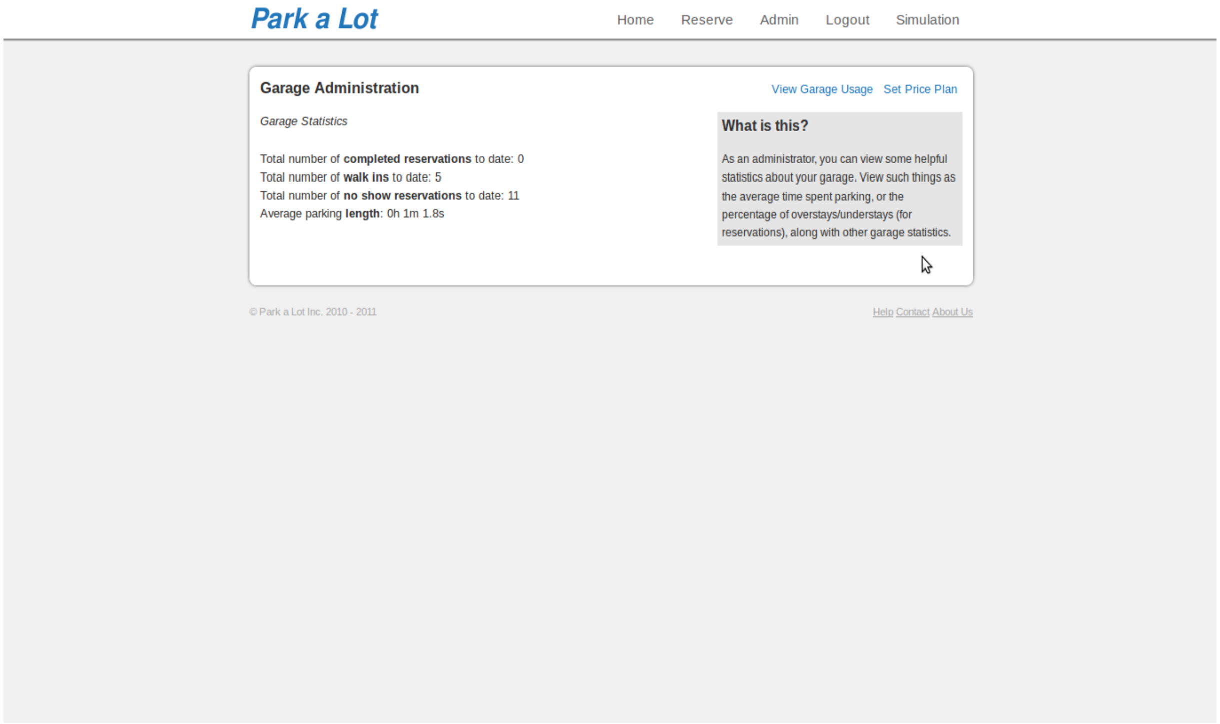
Are you sure you want to remove this vehicle?

License plate ABC-123 in the state of CT.

© Park a Lot Inc. 2010 - 2011

View Parking area Usage

From this page, **administrators** can view a few statistics about the parking area, such as the average time spent parking, the percentage of overstays/understays, the number of no-shows, etc.



Garage Administration

Garage Statistics

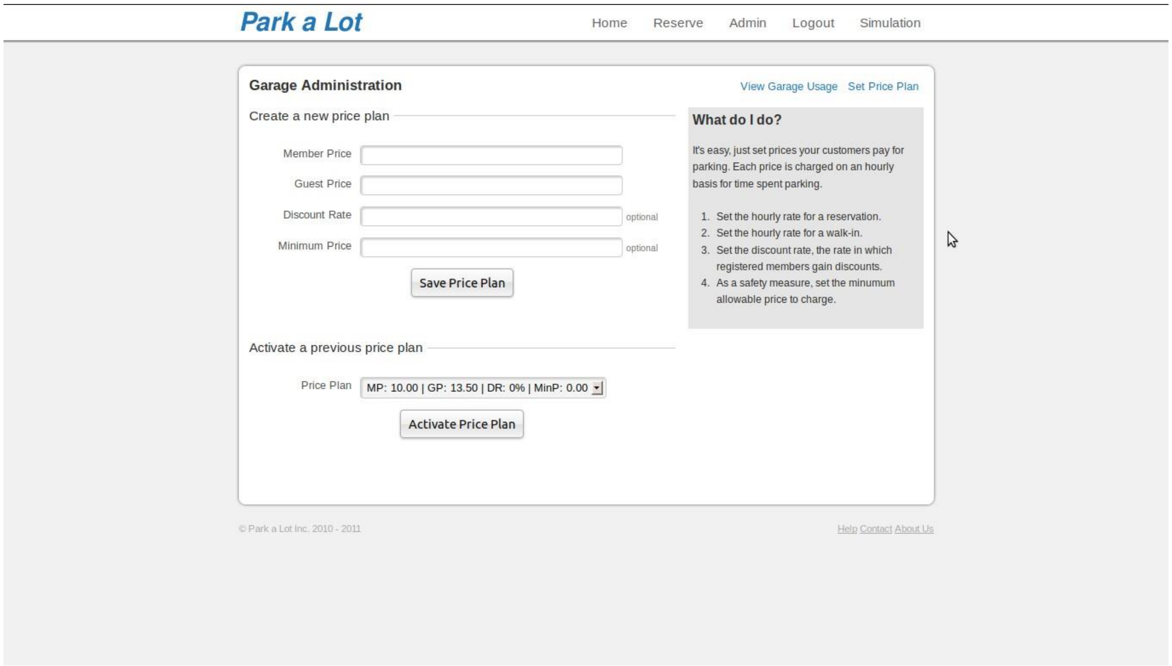
Total number of **completed reservations** to date: 0
Total number of **walk ins** to date: 5
Total number of **no show reservations** to date: 11
Average parking **length**: 0h 1m 1.8s

What is this?

As an administrator, you can view some helpful statistics about your garage. View such things as the average time spent parking, or the percentage of overstays/understays (for reservations), along with other garage statistics.

Set Price Plan

From this page, **administrators** can set new price plans, as well as activate old price plans.



Garage Administration

Create a new price plan

Member Price
Guest Price
Discount Rate optional
Minimum Price optional

Activate a previous price plan

Price Plan: MP: 10.00 | GP: 13.50 | DR: 0% | MinP: 0.00

What do I do?

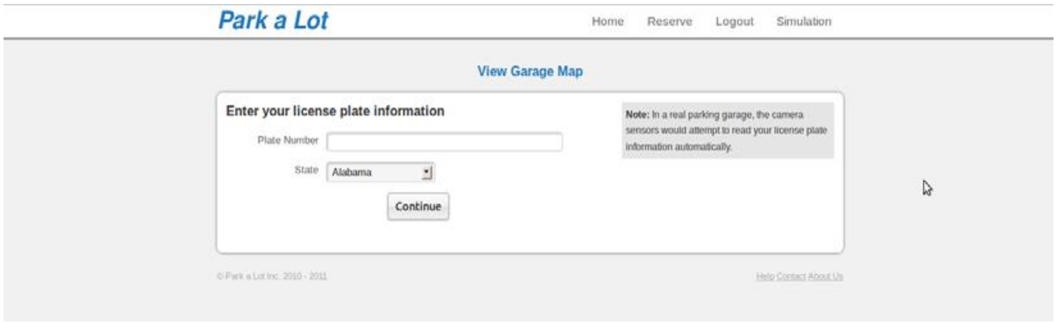
It's easy, just set prices your customers pay for parking. Each price is charged on an hourly basis for time spent parking.

1. Set the hourly rate for a reservation.
2. Set the hourly rate for a walk-in.
3. Set the discount rate, the rate in which registered members gain discounts.
4. As a safety measure, set the minumum allowable price to charge.

Simulation

To test our system, we implemented a simulation test bench. The simulation is to emulate as close as possible the process of physically parking in our parking area. Below are a few screenshots of our simulation.

Welcome Screen



Enter your license plate information

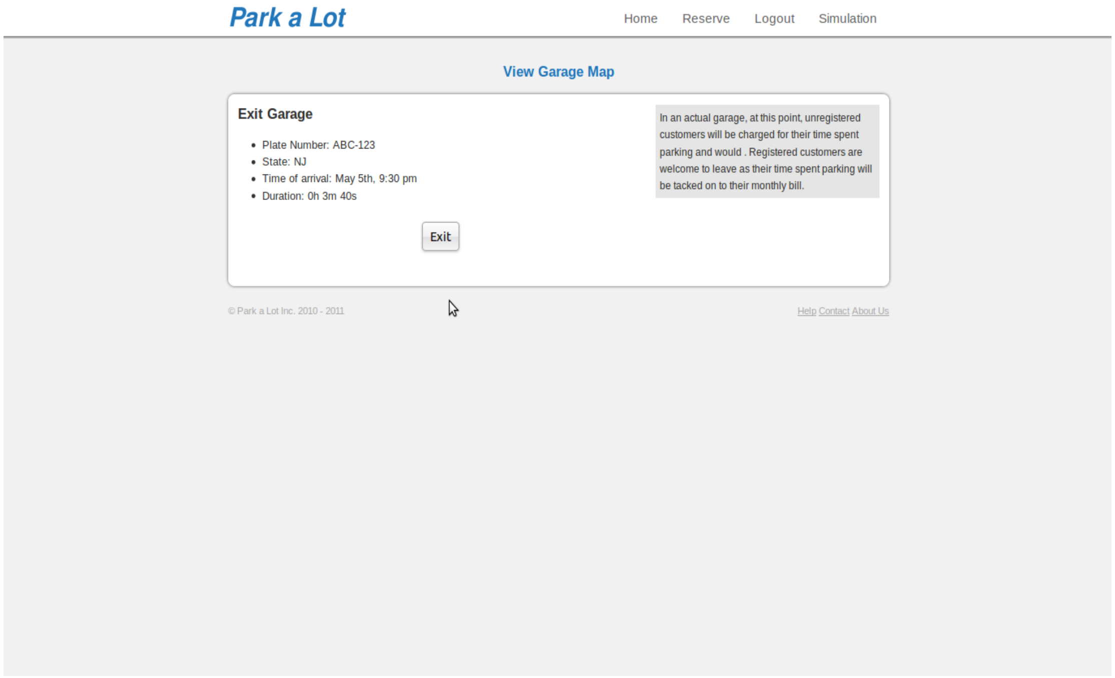
Plate Number

State: Alabama

Note: In a real parking garage, the camera sensors would attempt to read your license plate information automatically.

© Park a Lot Inc. 2010 - 2011

Exit Screen



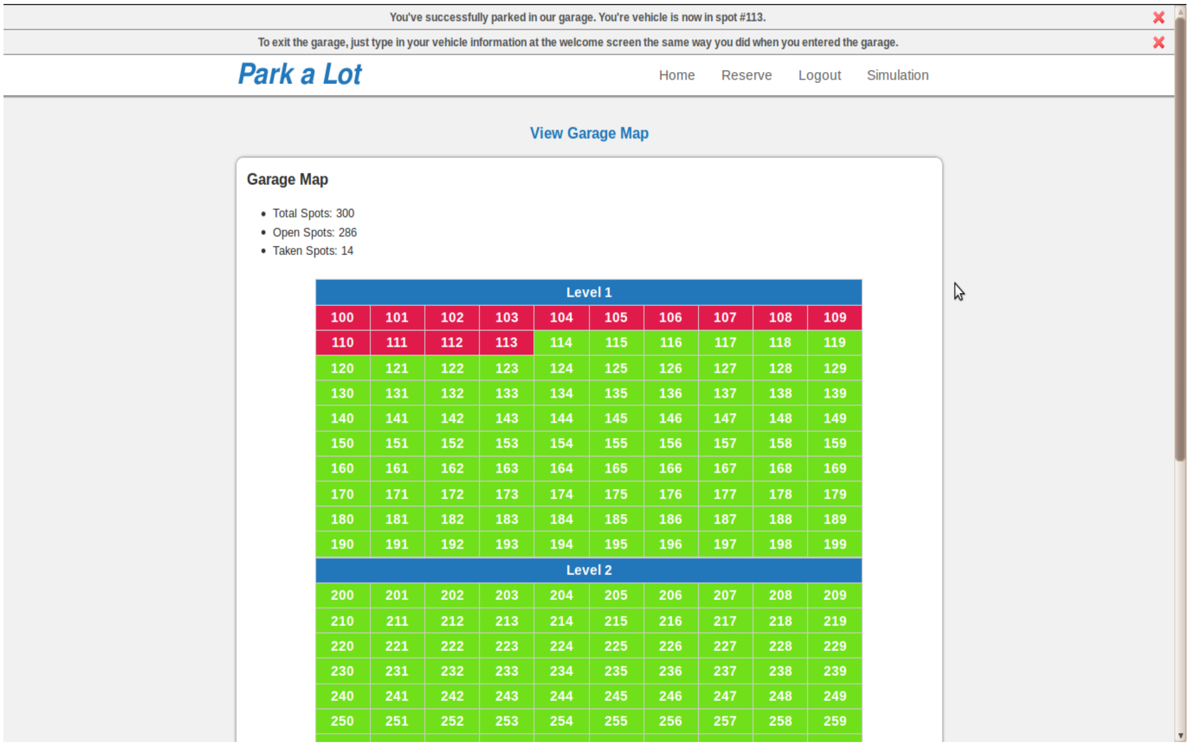
Exit Garage

- Plate Number: ABC-123
- State: NJ
- Time of arrival: May 5th, 9:30 pm
- Duration: 0h 3m 40s

In an actual garage, at this point, unregistered customers will be charged for their time spent parking and would . Registered customers are welcome to leave as their time spent parking will be tacked on to their monthly bill.

© Park a Lot Inc. 2010 - 2011

Parking area Map



You've successfully parked in our garage. You're vehicle is now in spot #113.

To exit the garage, just type in your vehicle information at the welcome screen the same way you did when you entered the garage.

View Garage Map

Garage Map

- Total Spots: 300
- Open Spots: 286
- Taken Spots: 14

Level 1									
100	101	102	103	104	105	106	107	108	109
110	111	112	113	114	115	116	117	118	119
120	121	122	123	124	125	126	127	128	129
130	131	132	133	134	135	136	137	138	139
140	141	142	143	144	145	146	147	148	149
150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169
170	171	172	173	174	175	176	177	178	179
180	181	182	183	184	185	186	187	188	189
190	191	192	193	194	195	196	197	198	199
Level 2									
200	201	202	203	204	205	206	207	208	209
210	211	212	213	214	215	216	217	218	219
220	221	222	223	224	225	226	227	228	229
230	231	232	233	234	235	236	237	238	239
240	241	242	243	244	245	246	247	248	249
250	251	252	253	254	255	256	257	258	259

Our entire user interface is aimed at an ease-of-use for the user. We've accomplished this by limiting the number of options on each page to the minimal, only the necessary actions to be taken are shown on a page. Our longest form is our user registration form, which only requires 5 input text fields. Our average form only requires 3 input fields, making form completion very simple for the user. Our most complicated form, our create reservation form, is already filled in with the basic information, such as the start date and start time (which we take as starting 3 hours from the current time).

Conclusion and Future Work

This project was difficult from the beginning, mainly because none of our group members had ever worked on a full-scale software project like this before. We had all coded applications, but the planning aspect of it and the formulation of such length reports proved somewhat a challenge.

However, at the conclusion of the semester we now have a functioning software system in place, and a long list of documentation and design reports supporting it. Although it was not simple or easy to put together, our careful design process and extremely clean coding style helped to create a powerful and easy to understand piece of code