Requirements Document:
Car Rental Service

Group 4
CSCI 4050: Software Engineering
Instructor: Krys Kochut
Osama Mansour, Stephen Patton, Vincent Lee, and Minh Pham

Table of Contents

1. Introduction

1.1 Purpose of the system

The purpose of the project is to develop a good online car rental service for young drivers. The car company uses very flexible rental agreements so that both people under age 25 and drivers older than 25 can rent. The company will be only available online. The system will have administrators with full access to the system. Customers can sign up, log in, search a car, reserve a car, pay rental fees, and terminate their accounts. The system allows multiple users to access and search for cars concurrently.

1.2 Scope of the System

The proposed system will be able to deal with all aspect of car rental and management. Users can check which car is available and at which location. Drivers can request a vehicle for a certain date and time to pick up at a specific location. If the requested car is not available at a desired location, the system can also check whether a similar car is available at an alternate location. The key stakeholders of the system are: customers and administrators, and their interest to the system are listed as below:

Stakeholders	Interests
Customers	Register his or her membership Search for a car and check the availability Reserve a car, pay the fee, and cancel the reservation Having a stable system, e.g. what if the system crashes while taking a reservation
Administrators	Manage the whole system, e.g. add more administrators, add more cars or another location

1.3 Objectives and Success Criteria

- The system should successfully allow an administrator to add, modify, delete, and search/ list entries of other administrators, rental locations, cars, and customers. They should be able to define and maintain a number of vehicle types, enter individual vehicles and specify their types, prices and other important properties. They should be able to remove a customer for serious violations of the rental agreement.
- The system should successfully allow an administrator to view the information and car rental history of all customers; he can also view all the car, availability and locations, and email to notify customers about reservation, late fee, cancellation, and answer some specific questions.
- The system should successfully allow a customer to create an account and pay the initial 6-month membership fee. It also allows customers to view all available cars at all locations, and to register for a car as well as cancel the reservation. If the requested vehicle is not available, the system can suggest a similar vehicle at a different location. The reservation must be specific in vehicle type, pickup time and the length of the rental.

The customer can also provide information about the condition of the returned vehicle and give comments about the vehicle and the rental service if desired. Finally, they can renew the membership as well as terminate their account**.**

1.4 References

- The project will be in Java or C++ specification
- Persistent data store with MySQL
- The system use accepted standards (HTML,SQL, …)
- The system must be accessible from a common Web browser (such as Mozilla Firefox, Google Chrome and Microsoft Internet Explorer)

1.5 Overview

The system being developed will be an easy to use online reservation system which supports all the requirements described in this document. It also will be capable to maintain data integrity while allowing a large number of drivers to access concurrently.

2. Proposed System

2.1 Overview

The proposed system uses flexible rental agreements and allows young and old drivers to rent a car as long as they can confirm their driving credentials. It will be available online. It will allow drivers to search, register a vehicle, cancel a reservation, and pay fees. Administrators will be allowed to manage the whole system, such as information regarding drivers, vehicles, locations, and ensure the stability of the system. Also, the system can be accessed by a large amount of users (administrators and customers) at the same time. The system provides an easy-to-use interface that is compatible to most of web browsers.

2.2 Functional Requirements

- Administrator should be able to add, remove, and edit all vehicle information, customer information, and business pricing information as needed.
- Customers should be able to search for and reserve available cars and should be able to edit their personal information, credit card information, and rental history. Customers should be able to cancel reservations within a certain time period, access billing information for past and present rentals, return a vehicle, and terminate membership through the car rental service system at any time.

2.3 Non-Functional Requirements

2.3.1 Performance

- The system should be provide the ability to modify, delete, search, and add users, vehicles, and locations in-line with real world transactions.
- Allow for multi-user access with no noticeable delay.

2.3.2 Reliability

- The system should work 24hrs a day, seven days a week.
- Maintain as close to 100% uptime SLA (service level agreement). Maintenance included, the system should have near 99.99% uptime
- The system will allow administrators access to customer accounts when they provide a name and driver's license

2.3.3 Maintainability

- The system should allow for system administrators should be able to make changes to any of the information currently stored in the system.
- The system must use a persistent data store for all the relevant data.
- Databases will make nightly backups of all data.

2.3.4 Implementation

- The system back-end should be created using Java or C++ with a MySQL database.
- The system front-end should be created using accepted standards including HTML, CSS, Jquery, etc.
- The system should work on the most popular current major browsers.

2.3.5 Extensibility & Expandability

- The system should be developed in a way to accommodate future functionality and expansion into variety of platforms.

2.4 System Model

2.4.1 Scenarios

Scenario name	**administratorAddVehicleType**
Participating actor Instances	John: Administrator
Flow of events	1. John would like to add a new vehicle type to system. John logs into the system and selects the vehicles page. 2. John selects the add vehicle type button and fills in the required fields in the provided form: name, price, etc. John clicks continue. 3. The system displays the new vehicle type with provided information. Once John is satisfied with this new vehicle type he clicks save to create the new vehicle type.

Scenario name	**administratorRemoveVehicleType**
Participating actor Instances	John: Administrator
Flow of events	1. John would like to remove a poorly performing vehicle type. John logs into the systems and selects the vehicles page. 2. John selects the vehicle type to be removed in a drop down box and clicks remove. 3. The system displays a page listing the vehicle type to be removed and the current vehicles that will be no longer available. Once John is satisfied with these changes, he clicks save and the changes become effective.

Scenario name	**administratorSetRentalPrice**
Participating actor Instances	John: Administrator
Flow of events	1. John wants to set the rental price of a vehicle type. Once logged into the system, he navigates to the pricing page and selects vehicles. 2. John chooses the vehicle type and is presented with a form containing the current price of hourly and daily rentals. John enters in new rates for one or both of the fields. John clicks continue. 3. The system displays a confirmation screen with the current vehicle pricing and the new pricing John set. John reviews these changes and clicks save to initiate the changes.

Scenario name	**administratorSetMembershipPrice**
Participating actor Instances	John: Administrator

Flow of events	1. John would like to change the price of the membership tiers. After logging in, he navigates to the pricing page and selects memberships. 2. John chooses the tier he would like to update pricing on, and is presented with a form containing the current price. John enters in a new price and clicks continue. 3. The system displays a confirmation screen with the current tier's price and the new pricing John has set. John reviews these changes and clicks save to activate the changes.

Scenario name	**administratorAddRentalLocation**
Participating actor Instances	John: Administrator
Flow of events	1. John is expanding his business and would like to add a new rental location. Once logged in, he selects the locations page. 2. John then chooses add rental location. He is presented with a form containing: name, address, vehicle capacity, etc. John fills in the required information and clicks continue. 3. The system displays a confirmation screen with the new locations information. John reviews the information. Once he is satisfied, he clicks save to create the new rental location.

Scenario name	**administratorModifyRentalLocation**
Participating actor Instances	John: Administrator
Flow of events	1. John needs to modify a rental location's information because his landlord's new contract cuts his lot substantially. John logs into the system and visits the rental location page. 2. John selects the rental location through a dropdown box and clicks edit. He is presented with the location's property information in a form. He modifies the appropriate values, and clicks continue. 3. The system displays a confirmation screen showing location's old properties and the new ones. Once John is satisfied with these changes he clicks save to make them effective.

Scenario name	**administratorRemoveRentalLocation**
Participating actor Instances	John: Administrator

Flow of events	1. John is notified by a landlord that the contract for a particular rental location will not be renewed. John logs into the system and navigates to the rental location page. 2. John selects the rental location to be removed from a drop down list, and clicks remove. 3. The system displays a confirmation screen showing the location to be removed. Once John has verified that the location to be removed is correct, he clicks save to remove the rental location.

Scenario name	**administratorAddVehicle**
Participating actor Instances	John: Administrator
Flow of events	1. John recently purchased additional vehicles and they have been prepared at the respected rental locations. John would like to add them to the system, and allow customers to begin renting them out. 2. John logs in and navigates to the vehicles page. John clicks the add vehicle button. He is presented with a form containing: vehicle type, make, model, year, registration tag, current mileage, date last serviced, condition, etc. John fills out this required information and clicks continue. 3. The system displays a confirmation screen showing the new vehicle and its information. John reviews this information. Once verified, he clicks save to add the new vehicle to the fleet.

Scenario name	**administratorModifyVehicle**
Participating actor Instances	John: Administrator
Flow of events	1. John just got a vehicle serviced, and needs to add this information to the vehicle records. He logs into the system, and selects the vehicles page. 2. John searches for the vehicle through a search bar, or locates it through its current location. John clicks modify on the vehicle profile, and is presented with a forum containing the vehicle information. He updates the last serviced date, and clicks continue. 3. The system displays a confirmation screen showing the changes made to the vehicle. John reviews these changes, and once verified, he clicks save to update the vehicle’s information.

Scenario name	**administratorRemoveVehicle**
Participating actor Instances	John: Administrator

Flow of events	1. John finished a performance review, and found a vehicle that is unprofitable and would like to remove it from the system. He logs in and navigates to the vehicles page. 2. John searches for the vehicle using a search bar and clicks modify on the vehicle profile. He is then presented with a form with the vehicle's information. At the bottom there is a button to remove the vehicle. He clicks the remove button. 3. The system displays a confirmation screen showing the vehicle to be removed. John reviews these changes, and clicks save to remove the vehicle from the fleet.

Scenario name	**administratorTerminateUser**
Participating actor Instances	John: Administrator
Flow of events	1. John finds out that a customer damaged a vehicle and failed to report the damage. 2. John logs into the site and navigates to customers and searches for the customer name. Once a match is located he clicks modify. He is brought to a form containing the users profile information. At the bottom he clicks disable user. 3. The system displays a confirmation screen showing the user and information that is going to be disabled. Once this information is verified, John clicks save, and the user can no longer use the service.

Scenario name	**administratorModifyCustomerInformation**
Participating actor Instances	John: Administrator
Flow of events	1. John needs to help a customer modify some information on their account. The customer says the system is not working. John logs into the system and navigates to the customer page. 2. John locates the customer profile and verifies the customer through information on file. John clicks on the edit tab on the profile and is directed to a form with the customer's information. He modifies the requested fields, and clicks continue. 3. The system displays a confirmation screen with the previous information and information updated. Once John verifies the changes are correct, he clicks save to employ the changes.

Scenario name	**administratorAddAdministrator**
Participating actor Instances	John: Administrator

Flow of events	1. John would like to create an administrator account for his business partner. He logs into the system, and navigates to the administrator page. 2. John clicks the add additional administrator. He is presented with a form with fields: name, username, temporary password, etc. John fills out the required information and clicks the continue button. 3. The system displays a confirmation screen showing the administrator to be added to the system. Once John verifies the information, he clicks create to add the new administrator.

Scenario name	**customerLogIn**
Participating actor Instances	Paul: Customer
Flow of events	1. Paul opens the web page and types in his username and password. 2. The system loads the customer home page and displays it to Paul.

Scenario name	**customerLogOut**
Participating actor Instances	Paul: Customer
Flow of events	1. Paul is finished using the website, locates the logout button at the top of the page, and clicks on it. 2. The system returns to the login screen.

Scenario name	**customerRegisterNewID**
Participating actor Instances	Paul: Customer
Flow of events	1. Paul opens the webpage and clicks on the register new user link 2. The system loads the registration page which will prompt Paul for a username, password, full name, email, home address, phone number, driver's license number, and credit card number. It will also ask Paul to pay the membership fee. 3. Paul will enter all of the fields, agree to pay the fee, and click the register button 4. The system will add Paul to its database
Scenario name	**customerRenewMembership**
Participating actor Instances	Paul: Customer

Flow of events	1. Paul clicks on the “Renew/Extend Membership Fee” link on the 'My Account' page 2. Paul selects how long he wants to extend his membership. 3. The system asks Paul to confirm his purchase and Paul clicks on confirm 4. The system updates Paul's information and prints a confirmation message

Scenario name	**customerTerminateMembership**
Participating actor Instances	Paul: Customer
Flow of events	1. Paul clicks on the 'Cancel Membership' link on the 'My Account' page 2. The system displays the following message, “Once you cancel your membership, it cannot be undone and no money will be refunded. If you plan on using this service in the future you will have to make a new account. Are you sure you want to cancel your membership?” and prompts the user with a yes and no option 3. Paul clicks on 'yes'. 4. The system deletes Paul from the database

Scenario name	**customerModifyName**
Participating actor Instances	Paul: Customer
Flow of events	1. Paul clicks on the 'Update Information' link on the 'My Account' page 2. The system prompts Paul with the following prompt: “Which do you wish to modify?” and lists several options including 'Name' 3. Paul clicks on the 'Name' option 4. The system displays his current name as it is in the system along with a cancel button if Paul changes his mind. It also provides a space for Paul to enter his new name and a space for him to confirm his name 5. Paul enters his name and clicks the 'Update Name' link 6. The system updates Paul's name and displays a confirmation message

Scenario name	**customerModifyPhoneNumber**
Participating actor Instances	Paul: Customer

Flow of events	1. Paul clicks on the 'Update Information' link on the 'My Account' page 2. The system prompts Paul with the following prompt: “Which do you wish to modify?” and lists several options including 'Phone Number' 3. Paul clicks on the 'Phone Number' option 4. The system displays his current phone number as it is in the system along with a cancel button if Paul changes his mind. It also provides a space for Paul to enter his new phone number and a space for him to confirm his phone number 5. Paul enters his name and clicks the 'Update Phone Number' link 6. The system updates Paul's phone number and displays a confirmation message

Scenario name	**customerModifyAddress**
Participating actor Instances	Paul: Customer
Flow of events	1. Paul clicks on the 'Update Information' link on the 'My Account' page 2. The system prompts Paul with the following prompt: “Which do you wish to modify?” and lists several options including 'Address' 3. Paul clicks on the 'Address' option 4. The system displays his current address as it is in the system along with a cancel button if Paul changes his mind. It also provides a space for Paul to enter his new address 5. Paul enters his name and clicks the 'Update Address' link 6. The system updates Paul's address and displays a confirmation message

Scenario name	**customerModifyDriversLicense**
Participating actor Instances	Paul: Customer

Flow of events	1. Paul clicks on the 'Update Information' link on the 'My Account' page 2. The system prompts Paul with the following prompt: “Which do you wish to modify?” and lists several options including 'Driver's License Information' 3. Paul clicks on the 'Driver's License Information' option 4. The system displays his current driver's license information as it is in the system along with a cancel button if Paul changes his mind. It also provides a space for Paul to enter his new driver's license information. 5. Paul enters his Information and clicks the 'Update Driver's License Number' link 6. The system updates Paul's driver's license number and displays a confirmation message

Scenario name	**customerModifyEmail**
Participating actor Instances	Paul: Customer
Flow of events	1. Paul clicks on the 'Update Information' link on the 'My Account' page 2. The system prompts Paul with the following prompt: “Which do you wish to modify?” and lists several options including 'Email' 3. Paul clicks on the 'Email' option 4. The system prompts Paul to type in his current email, his new email, and a confirmation of his new email 5. Paul enters his information and clicks the 'Update Email link 6. The system updates Paul's email and displays a confirmation message

Scenario name	**customerModifyCreditCard**
Participating actor Instances	Paul: Customer

Flow of events	1. Paul clicks on the 'Update Information' link on the 'My Account' page 2. The system prompts Paul with the following prompt: “Which do you wish to modify?” and lists several options including 'Credit Card' 3. Paul clicks on the 'Credit Card' option 4. The system asks for Paul's new credit card information including: 16 digit number, expiration date, full name, billing address, and security code 5. Paul enters the information and clicks on 'Submit' 6. The system updates Paul's information and displays a confirmation message

Scenario name	**customerModifyPassword**
Participating actor Instances	Paul: Customer
Flow of events	1. Paul clicks on the 'Update Information' link on the 'My Account' page 2. The system prompts Paul with the following prompt: “Which do you wish to modify?” and lists several options including 'Password' 3. Paul clicks on the 'Password' option 4. The system prompts Paul to type in his current password, his new password, and a confirmation of his new password 5. Paul enters his information and clicks the 'Update Password' link 6. The system updates Paul's password and displays a confirmation message

Scenario name	**customerReserveVehicle**

Participating actor Instances	Paul: Customer
Flow of events	1. Paul is on the page of a vehicle he wants to reserve and clicks on the 'Reserve Vehicle' link 2. The system loads a page that prompts Paul for the date, time, duration, and drop-off destination for the rental of the vehicle. 3. Paul fills in the required information and clicks submit. 4. The system displays the following information: • Rental site • Car information • Date and time of pick up and drop off • Drop-off destination • Price The system asks Paul if this information is correct and to confirm payment 5. Paul confirms the information by clicking 'Confirm' 6. The system updates that Paul has made this reservation and displays a message confirming the reservation

Scenario name	**customerCancelReservation**
Participating actor Instances	Paul: Customer
Flow of events	1. Paul clicks on the 'Cancel Reservation' link in the “My Account” page 2. The system displays all the reservations currently on Paul's account and asks him which one he would like to cancel 3. Paul selects the reservation he wishes to cancel 4. The system asks Paul if he is sure and if it is less than an hour before the reservation time reminds Paul that there will be a fee charged to his account for late cancellation. 5. Paul clicks yes that he is sure 6. The system updates the information and displays a confirmation message

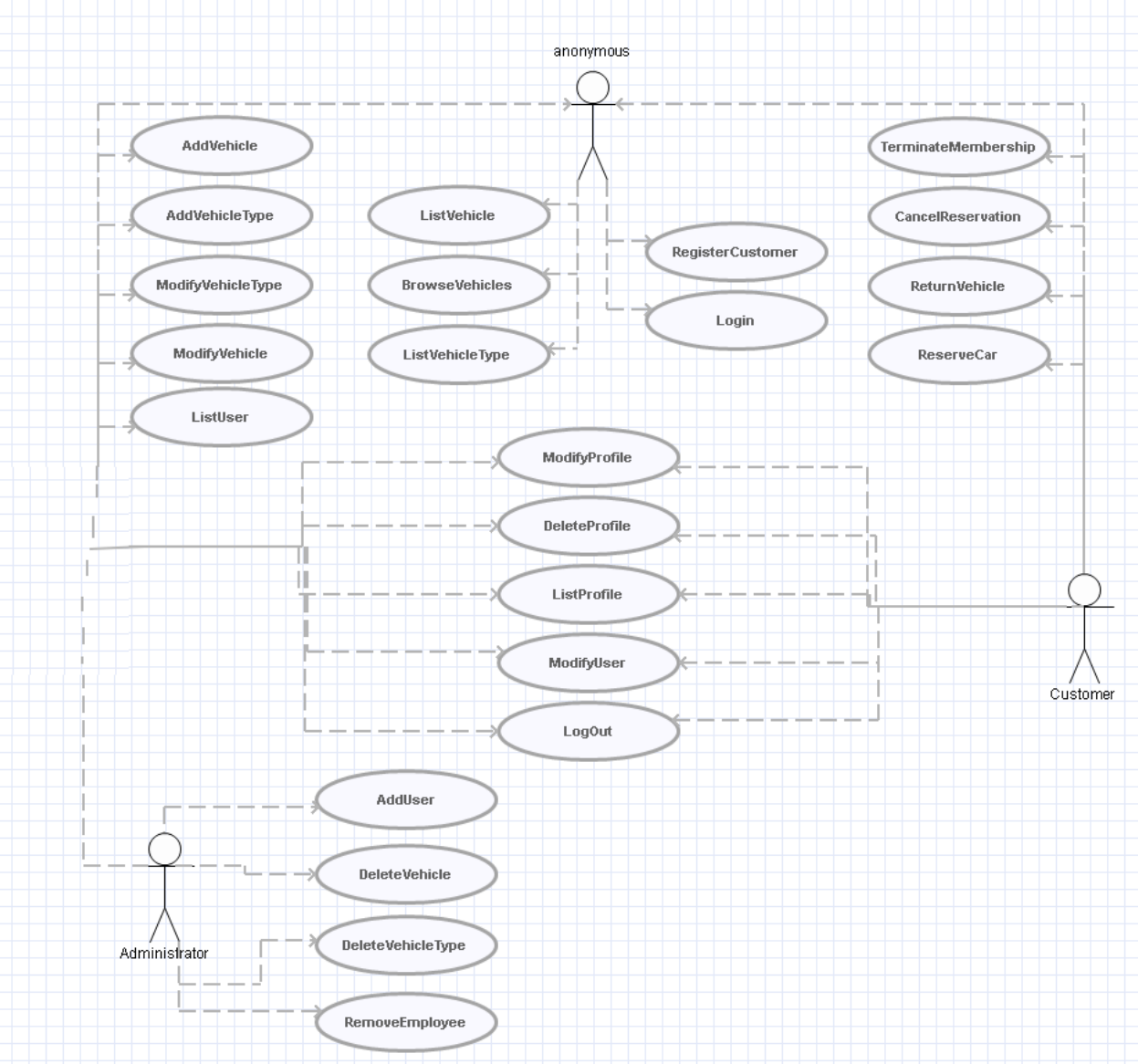
Scenario name	**customerNotifyVehicleReturn**

Participating actor Instances	Paul: Customer
Flow of events	1. Paul clicks on the 'Notify Vehicle Return' Link on the 'My Account' page 2. The system displays the Vehicle information and return site and prompts Paul to verify that it is correct 3. Paul clicks on yes 4. The system updates the return and displays a confirmation message

Scenario name	**customerBrowseVehicle**
Participating actor Instances	Paul: Customer
Flow of events	1. At the home screen Paul clicks on browse vehicles 2. The system provides the following optional constraints to Paul: • Type of vehicle • Location • Date and time of pick up and drop off • Price range 3. Paul chooses the constraints that he wants and clicks browse 4. If there are cars available that match Paul's constraints, the system displays them. It will display the car information, location, dates and times available, and price.

Scenario name	**customerViewRentalHistory**
Participating actor Instances	Paul: Customer
Flow of events	1. From the 'My Account' page, Paul clicks on 'View Rental History'. 2. The system displays all previous rentals and any current reservations

2.4.2 Use Case Model



2.4.2.1 ReturnVehicle

Name	Return Vehicle

ID	ReturnVehicle
Version	1.0
Author	Stephen Patton
Date	9/8/13
Summary	Use case for customers notifying the system that they have returned the car they have rented
Basic Path	1. The system displays various information on the home page, including a link to the customer's account. The customer clicks on the link to his/her account 2. The system loads the account page. On the account page is a link that reads “Return Vehicle.” The customer clicks on the Return Vehicle. 3. The system prints a confirmation message asking if the customer is sure, and waits for the customer to confirm by clicking on the confirm button. 4. The customer clicks the confirm button and the system updates the cars being available. The system will then bring the customer back to the home page
Alternative Paths	1. In step 3, if the customer is late in returning the vehicle, the system will in addition to asking the user to confirm that the car is being returned, inform the customer of the additional money that is owed as a result of returning the car late. 2. In step 3, if the customer clicks cancel instead of confirm, he/she will be brought back to the account settings page.
Exception Paths	1. In step 2, the customer clicks on “Return Vehicle” when he/she has no vehicle currently being rented. The system will tell the customer that he/she has no vehicle currently rented and return to the account settings page
Extension Points	None
Triggers	A customer wants to return a vehicle
Assumption	None
Pre-conditions	The customer returns the car to the car rental site specified.
Post-conditions	The customer's payment is received, including possible late payments, and the car is updated to be available for other customers.

2.4.2.2 BrowseVehicles

Name	Browse Vehicles
ID	BrowseVehicles
Version	1.0
Author	Stephen Patton
Date	9/9/13
Summary	Use Case for browsing the Vehicles in the database through the website.
Basic Path	1. At the home page, a user clicks on the Browse button. 2. The system loads a page with options to limit browsing for a vehicle, including vehicle type, pick-up location, price range, vehicle make and model, and date and time available. 3. The user selects the limitations he/she wants and clicks on browse. 4. The system finds every vehicle in the database that falls within the search limitations and displays them on the screen
Alternative Paths	1. In step 4, if the user decides that he/she wants to narrow the search parameters, he/she can update them and the system will once again search the database and print the updated car list.
Exception Paths	1. If in step 4, the system can not find a single vehicle that matches the search, it will instead print that there are no vehicles that match that description. It will then give the user the option of making another search
Extension Points	None
Triggers	The user wants to browse available cars
Assumption	None
Pre-conditions	There are available cars
Post-conditions	The user can see the available cars within the parameters he/she has chosen.

2.4.2.3 RegisterCustomer

Name	Register Customer
ID	RegisterCustomer
Version	1.0
Author	Stephen Patton
Date	9/9/13
Summary	A new customer user registers with the system
Basic Path	1. The user clicks on the register a new account button 2. The system loads the registration page. There is a form asking for the following information: full name, phone number, address, email, driver's license information, and credit card information. 3. The user fills out the information and clicks continue 4. The system informs the user that there will be a membership fee and prompts the user to agree to pay. 5. The user clicks accept 6. The user is now registered with the system
Alternative Paths	1. In step 3, one or more of the information fields are incomplete or incorrect. The system asks the user to correct the information. The user corrects the information and moves on to step 4.
Exception Paths	1. In step 5, the user does not agree to pay the fee. The registration is terminated and the user is brought back to the home page
Extension Points	None
Triggers	A customer wants to register with the system
Assumption	The driver's license and credit card information are confirmed to be correct
Pre-conditions	None
Post-conditions	The database is updated to include this new user

2.4.2.4 AddVehicle

Name	Add Vehicle
ID	AddVehicle
Version	1.0
Author	Vincent Lee
Date	9/8/2013
Summary	Use case for an administrator to add a vehicle to the fleet of existing vehicles.
Basic Path	1. The system provides a vehicles page listing the number of actions that the administrator can take affecting the fleet of vehicles, after the administrator is logged into the system. 2. The administrator indicates that a new vehicle is to be added to the system. 3. The system prompts the administrator for: a. The vehicle type of the new vehicle b. The make of the new vehicle c. The model of the new vehicle d. The year of the new vehicle e. The registration tag of the new vehicle f. The current mileage of the new vehicle g. The time of last servicing of the new vehicle h. The condition of the new vehicle i. The rental location assigned to the new vehicle The system also prompts the administrator for action to take after the information is entered: a. Continue to review changes before saving to database b. Cancel current vehicle add 4. The administrator clicks the save button in step 3, and the system displays a confirmation page with the entered information and prompts the administrator to: a. Add Vehicle and return to the vehicle page b. Add Vehicle & Another c. Edit vehicle information d. Cancel current vehicle add 5. The administrator chooses Add Vehicle and returns to the vehicle page, and the system stores the record for the new vehicle to the database.
Alternative Paths	1. In step 4, if the administrator clicks the Edit button, the use case returns to step 3, with all of the information in the forms retained to allow necessary changes. 2. In step 3 and 4, if the administrator clicks the Cancel button, the use case terminates, and the system returns to the vehicle page in step 1.

Exception Paths	1. In step 3, the system determines that a vehicle with the given registration tag already exists after Continue button is pressed, the system displays “A vehicle with registration tag *** already exists” message and the system stays at step 3. 2. In step 3, the system determines that a rental location is already at its maximum vehicle capacity after Continue button is pressed, the system displays “The rental location is at maximum capacity” message and the system stays at step 3.
Extension Points	1. In step 4, if the administrator clicks Add Vehicle and Another button, the system adds the vehicle to the system, and returns to step 3 in the basic path.
Triggers	A user with the administrator role is allowed to add a new vehicle to the system.
Assumption	None
Pre-conditions	The administrator is currently logged in and the session has not expired.
Post-conditions	Information is appropriately added to the database of the system.

2.4.2.5 ListVehicle

Name	List Vehicle
ID	ListVehicle
Version	1.0
Author	Vincent Lee
Date	9/8/2013
Summary	Use case for listing all vehicles in the fleet of vehicles, given a set of conditions.
Basic Path	1. The system provides a search box which allows searching of vehicle information. 2. The system prompts a user to enter in a keyword to search the vehicles in the database. A Search button is located next to the search box. 3. The users fills in the search field and clicks the Search button. 4. The system searches through the database, and returns the result list to the user.
Alternative Paths	None
Exception Paths	1. If no vehicle are found, the system informs the user
Extension Points	None
Triggers	A user elects to list some kind of vehicle in the system.
Assumption	None
Pre-conditions	None
Post-conditions	The system displays the user with what he or she wants to view, and nothing changes to the underlying data in the system.

2.4.2.6 ModifyVehicle

Name	Modify Vehicle
ID	ModifyVehicle
Version	1.0
Author	Vincent Lee
Date	9/8/2013
Summary	Use case for an administrator to modify the information of a vehicle.
Basic Path	1. The system provides a vehicles page listing the number of actions that the administrator can take affecting the fleet of vehicles, after the administrator is logged into the system. 2. The system implements the ListVehicle use case to locate the vehicle. 3. Once the vehicle is located, the administrator clicks on the Edit button of the vehicle. 4. The system displays a vehicle profile page containing the information of the vehicle. 5. The administrator makes the necessary modifications to the vehicle profile. The screen includes buttons: Continue and Cancel. 6. The administrator clicks the Continue button and is directed to a confirmation page with the changes listed. The screen includes buttons: Save, Cancel, and Edit. 7. The administrator clicks the Save button and changes are saved to the database.
Alternative Paths	1. In step 5 and 6, if the administrator clicks the Cancel button, the use case terminates, and the system returns to the vehicle page in step 1.
Exception Paths	1. In step 6, if the administrator clicks the Edit button, the use case returns to step 4, with all of the information in the forms retained to allow necessary changes.
Extension Points	None
Triggers	An administrator elects to modify a vehicle’s information.
Assumption	None
Pre-conditions	The administrator is currently logged in and the session has not expired.
Post-conditions	The modified information is available to other use cases. The attributes of the vehicle changed are appropriately updated in the database.

2.4.2.7 DeleteVehicle

Name	Delete Vehicle
ID	DeleteVehicle
Version	1.0
Author	Vincent Lee
Date	9/8/2013
Summary	Use case for an administrator to remove a vehicle from the fleet of existing vehicles.
Basic Path	1. The system provides a vehicles page listing the number of actions that the administrator can take affecting the fleet of vehicles, after the administrator is logged into the system. 2. The system implements the ListVehicle use case to locate the vehicle. 3. Once the vehicle is located, the administrator clicks on the Edit button of the vehicle. 4. The system displays a vehicle profile page containing the information of the vehicle. The screen contains the Delete button at the bottom. 5. The administrator clicks the Delete button. 6. The system displays a confirmation screen containing the information of the vehicle to be deleted. The screen has buttons: Delete and Cancel. 7. An administrator clicks the Delete button and the system displays a "Successfully deleted" message, and returns to the main vehicle page.
Alternative Paths	1. In step 6, if the administrator clicks the Cancel button, the use case terminates, and the system returns to the vehicle page in step 1.
Exception Paths	None
Extension Points	None
Triggers	A user with administrator role elects to delete a vehicle from the system.
Assumption	None
Pre-conditions	The administrator is currently logged in and the session has not expired.
Post-conditions	The records concerning the vehicle the administrator wants to remove are deleted from the database.

2.4.2.8 AddVehicleType

Name	Add Vehicle Type
ID	AddVehicleType
Version	1.0
Author	Vincent Lee
Date	9/8/2013
Summary	Use case for an administrator to add a vehicle type to the existing vehicles types available.
Basic Path	1. The system provides a vehicles page listing the number of actions that the administrator can take affecting the fleet of vehicles, after the administrator is logged into the system. 2. The administrator indicates that a new vehicle type is to be added to the system. 3. The system prompts the administrator for: a. The name of the new vehicle type b. The hourly rental price of the new vehicle type c. The daily rental price of the new vehicle type The system also prompts the administrator for action to take after the information is entered: a. Continue to review changes before saving to database b. Cancel current vehicle type add 4. The administrator clicks the save button in step 3, and the system displays a confirmation page with the entered information and prompts the administrator to: a. Add Vehicle Type and return to the vehicle page b. Add Vehicle Type & Another c. Edit vehicle type information d. Cancel current vehicle type add 5. The administrator chooses Add Vehicle Type and returns to the vehicle page, and the system stores the record for the new vehicle type to the database.
Alternative Paths	1. In step 4, if the administrator clicks the Edit button, the use case returns to step 3, with all of the information in the forms retained to allow necessary changes. 2. In step 3 and 4, if the administrator clicks the Cancel button, the use case terminates, and the system returns to the vehicle page in step 1.
Exception Paths	1. In step 3, the system determines that a vehicle type with the given name already exists after Continue button is pressed, the system displays “The vehicle type already exists” message and the system stays at step 3.
Extension Points	1. In step 4, if the administrator clicks Add Vehicle Type and Another button, the system adds the vehicle type to the system, and returns to step 3 in the basic path.

Triggers	A user with the administrator role is allowed to add a new vehicle type to the system.
Assumption	None
Pre-conditions	The administrator is currently logged in and the session has not expired.
Post-conditions	Information is appropriately added to the database of the system.

2.4.2.9 ListVehicleType

Name	List Vehicle Type
ID	ListVehicleType
Version	1.0
Author	Vincent Lee
Date	9/8/2013
Summary	Use case for listing all vehicle types contained within the system, given a set of conditions.
Basic Path	1. The system provides a search box which allows searching of vehicle type. 2. The system prompts a user to enter in a keyword to search the vehicle types in the database. A Search button is located next to the search box. 3. The users fills in the search field and clicks the Search button. 4. The system searches through the database, and returns the result list to the user.
Alternative Paths	None
Exception Paths	None
Extension Points	None
Triggers	A user elects to list some kind of vehicle type in the system.
Assumption	None
Pre-conditions	None
Post-conditions	The system displays the user with what he or she wants to view, and nothing changes to the underlying data in the system.

2.4.2.10 ModifyVehicleType

Name	Modify Vehicle Type
ID	ModifyVehicleType
Version	1.0
Author	Vincent Lee
Date	9/8/2013
Summary	Use case for an administrator to modify the information in a vehicle type.
Basic Path	1. The system provides a vehicles page listing the number of actions that the administrator can take affecting the fleet of vehicles, after the administrator is logged into the system. 2. The ListVehicleType use case is implemented. 3. Once the vehicle type is located, the administrator clicks on the Edit button of the vehicle type. 4. The administrator makes the necessary modifications to the vehicle type. The screen includes buttons: Continue and Cancel. 5. The administrator clicks the Continue button and is directed to a confirmation page with the changes listed. The screen includes buttons: Save, Cancel, and Edit. 6. The administrator clicks the Save button and changes are saved to the database.
Alternative Paths	1. In step 4 and 5, if the administrator clicks the Cancel button, the use case terminates, and the system returns to the vehicle page in step 1.
Exception Paths	1. In step 5, if the administrator clicks the Edit button, the use case returns to step 4, with all of the information in the forms retained to allow necessary changes.
Extension Points	None
Triggers	An administrator elects to modify a vehicle type information.
Assumption	None
Pre-conditions	The administrator is currently logged in and the session has not expired.
Post-conditions	The modified information is available to other use cases. The attributes of the vehicle type changed are appropriately updated in the database.

2.4.2.11 DeleteVehicleType

Name	Delete Vehicle Type
ID	DeleteVehicleType
Version	1.0
Author	Vincent Lee
Date	9/8/2013
Summary	Use case for an administrator to delete a vehicle type from the list of vehicle types.
Basic Path	1. The system provides a vehicles page listing the number of actions that the administrator can take affecting the fleet of vehicles, after the administrator is logged into the system. 2. The system implements the ListVehicleType use case to locate the vehicle. 3. Once the vehicle type is located, the administrator clicks on the Edit button of the vehicle type. 4. The system displays a vehicle type profile page containing the information of the vehicle type. The screen contains the Delete button at the bottom. 5. The administrator clicks the Delete button. 6. The system displays a confirmation screen containing the information of the vehicle type to be deleted. The screen has buttons: Delete and Cancel. 7. An administrator clicks the Delete button and the system displays a "Successfully deleted" message, and returns to the main vehicle page.
Alternative Paths	1. In step 6, if the administrator clicks the Cancel button, the use case terminates, and the system returns to the vehicle page in step 1.
Exception Paths	None
Extension Points	None
Triggers	A user with administrator role elects to delete a vehicle type from the system.
Assumption	None
Pre-conditions	The administrator is currently logged in and the session has not expired.
Post-conditions	The records concerning the vehicle type the administrator wants to remove are deleted from the database.

2.4.2.12 ListProfile

Name	List Profile
ID	ListProfile
Version	1.0
Author	Vincent Lee
Date	9/8/2013
Summary	Use case for listing all user information in the system, given a set of conditions.
Basic Path	1. The system provides a search box which allows searching of profiles. 2. The system prompts a user to enter in a keyword to search the profiles in the database. A Search button is located next to the search box. 3. The users fills in the search field and clicks the Search button. 4. The system searches through the database, and returns the result list to the user.
Alternative Paths	None
Exception Paths	None
Extension Points	None
Triggers	A user elects to list some kind of profile in the system.
Assumption	Customers are only able to list their own profile.
Pre-conditions	None
Post-conditions	The system displays the user with what he or she wants to view, and nothing changes to the underlying data in the system.

2.4.2.13 ModifyProfile

Name	Modify Profile
ID	ModifyProfile
Version	1.0
Author	Vincent Lee
Date	9/8/2013
Summary	Use case for modifying the information contained in a user profile.
Basic Path	1. The system displays a user profile page listing the number of actions users can take based on their role. 2. The ListProfile use case is implemented. 3. Once the profile is located, the user clicks on the Edit button of the profile. 4. The user makes the necessary modifications to the profile. The screen includes buttons: Continue and Cancel. 5. The user clicks the Continue button and is directed to a confirmation page with the changes listed. The screen includes buttons: Save, Cancel, and Edit. 6. The user clicks the Save button and changes are saved to the database.
Alternative Paths	1. In step 4 and 5, if the user clicks the Cancel button, the use case terminates, and the system returns to the profile in step 1.
Exception Paths	1. In step 5, if the user clicks the Edit button, the use case returns to step 4, with all of the information in the forms retained to allow necessary changes.
Extension Points	None
Triggers	A user elects to modify a profile's information.
Assumption	Customers are only able to modify their own profile.
Pre-conditions	The user is currently logged in and the session has not expired.
Post-conditions	The modified information is available to other use cases. The attributes of the profile changed are appropriately updated in the database.

2.4.2.14 DeleteProfile

Name	Delete Profile
ID	DeleteProfile
Version	1.0
Author	Vincent Lee
Date	9/8/2013
Summary	Use case for deleting a user profile from the list of user profiles.
Basic Path	1. The system displays a user profile page listing the number of actions users can take based on their role. 2. The system implements the ListProfile use case to locate the profile. 3. Once the profile is located, the user clicks on the Edit button of the user profile. 4. The system displays a profile page containing the information of the user. The screen contains the Delete button at the bottom. 5. The user clicks the Delete button. 6. The system displays a confirmation screen containing the information of the user profile to be deleted. The screen has buttons: Delete and Cancel. 7. A user clicks the Delete button and the system displays a “Successfully deleted” message, and returns to the main profile page.
Alternative Paths	1. In step 6, if the user clicks the Cancel button, the use case terminates, and the system returns to the profile page in step 1.
Exception Paths	None
Extension Points	None
Triggers	A user with elects to delete a profile from the system.
Assumption	Customers are only able to delete their own profile.
Pre-conditions	The user is currently logged in and the session has not expired.
Post-conditions	The records concerning the profile the user wants to remove are deleted from the database.

2.4.2.15 Login

Name	Login
ID	Login
Version	1.0
Author	Osama Mansour
Date	9/6/13
Summary	Use case to login into the registration systems; Necessary to gain access to functionalities of the system
Basic Path	1. The system will display a screen prompting for user name and passwords. The login button will be displayed as well 2. A user provides a username and password and the login button. 3. The system locates the user matching the username and password. The system displays a customized navigation view dependent on the user information (Administrator, Customer).
Alternative Paths	None
Exception Paths	1. In step 2, if the user name and password are not located in the database, it returns an invalid user message.
Extension Points	None
Triggers	A user chooses to use the system
Assumption	None
Pre-conditions	The user is not already logged in
Post-conditions	The user session is available to other user cases accessible to the same user

2.4.2.16 LogOut

Name	Log Out
ID	LogOut
Version	1.0
Author	Osama Mansour
Date	9/6/13
Summary	Use case for a user to logout from the car rental service system, terminating the current user's session.
Basic Path	1. A user presses logout button located in the user's view. 2. The system will terminate the current User's session and display the confirmation of the logout process
Alternative Paths	None
Exception Paths	None
Extension Points	None
Triggers	A user elects to quit using the system and logout
Assumption	None

Pre-conditions	The user is currently logged inside the system and desires to exit. The user session is also intact
Post-conditions	The current session is terminated for this user.

2.4.2.17 AddUser

Name	Add an administrator
ID	AddUser
Version	1.0
Author	Osama Mansour
Date	9/6/13
Summary	Use case for adding an administrator to the car rental system.
Basic Path	1. The system will provide the administrator with a list of actions that can be chosen on their initial display page. 2. The administrator indicates a new administrator is to be added to the system. 3. The system will then prompt the administrator for a **name**, **ID**, and **password.** The system will prompt for an option to submit to the database or cancel. 4. The administrator will then be prompted to return to the main page or add another user. 5. The administrator returns to main page and the system updates the new employed user into the database.
Alternative Paths	1. In step 3 the administrator could have clicked the cancel button terminating the use case, and returning to the main page.
Exception Paths	1. In step 3 the system determines that a user with the given ID is already exists, and a “User exists” message will display
Extension Points	1. In step 4 the user could choose to add another user, the systems receives the submission, and returns to the main page
Triggers	A user with administrator role elects to add a new user to the system
Assumption	The administrator has not been suspended from adding users
Pre-conditions	The administrator is currently logged and the session is present.
Post-conditions	Information submitted is stored in the database correctly.

2.4.2.18 ListUser

Name	List User
ID	ListUser
Version	1.0
Author	Osama Mansour
Date	9/6/13
Summary	Use case for administrators to list users according to search settings
Basic Path	1. The system provides a main page listing number of actions that the administrator can take after the administrator logs into the system. 2. The system prompts the administrator for the search option from the list of actions. 3. The system prompts the administrator with a drop down list of roles administrator or customer 4. The system then prompts for two options. a. Start Search option to list the users b. a field to add a last name to the Search bar c. Cancel button back to main page 5. The administrator clicks the Start search option to list the users 6. The system searches through the database and returns all the results bounded by the search settings given.
Alternative Paths	1. In step 4, the administrator clicks Cancel button, returning to main page
Exception Paths	None
Extension Points	None
Triggers	An administrator desires to search a particular user
Assumption	None
Pre-conditions	The administrator is logged into the system with a current session present.
Post-conditions	The system displays the administrator the information requested through the search settings.

2.4.2.19 ModifyUser

Name	Modify User
ID	ModifyUser
Version	1.0
Author	Osama Mansour
Date	9/6/13
Summary	User case for User to modify User information within constraints placed by the system. a. Administrator may edit themselves and Customer information b. Customers may only edit themselves
Basic Path	1. The system provides a main page with a list of options one of which one option will allow user to modify his or her information. 2. The user will click the modify option and be directed to system display. 3. The user will then edit his or her information within the database by text field or drop box options designated by the system. 4. The user will hit the submit button. They system will display a confirmation window prompting the user to hit confirm or cancel updates. 5. The user will hit confirm. The system will locate the users information, and make the necessary updates to the database.
Alternative Paths	1. The admin will have the modified permissions described in the summary of the use case. Dependent on the options chosen the Basic path from step 3 2. In step 4 the user hits cancel returning the system to the initial state.
Exception Paths	None
Extension Points	None
Triggers	A user wants to edit his/her information or other user information within the system.
Assumption	The user has the correct permissions granted
Pre-conditions	The user is currently logged in to the system and the session has not expired.
Post-conditions	The user has made the updates to the database of the system with accordance to the access rights defined.

2.4.2.21 TerminateMembership

Name	Terminate Membership
ID	TerminateMembership
Version	1.0
Author	Osama Mansour
Date	9/6/13
Summary	Use case for a customer to terminate their member ship from within the car rental service system.
Basic Path	1. The system provides a list of actions displayed on the main page of the customer view one of which includes accounting settings. 2. The customer will select the account settings button and be redirected to a screen with various account information including name, address, license, and activity status. Below the displayed information will be an account termination button which customer will click. 3. The system will then display a prompt confirming the initiation of the termination of the membership with two options. a. Confirm Termination of Account b. Cancel Termination of account 4. The customer will choose the termination of the account. The customer will then be automatically be given a message by the system that they have been removed from the car rental system. 5. The system will log the user out of the database, and terminate the session.
Alternative Paths	1. In step 4, choosing cancel termination of account will exit the user from the termination screen back to the main customer view page.
Exception Paths	None
Extension Points	None
Triggers	A user desires to terminate his or her account
Assumption	They are a customer not an administrator. These will go through different processes stated in a use case
Pre-conditions	The user is logged into the system with a current active session
Post-conditions	The user is removed from all databases including information about the customer. The customer car rental history is preserved for business preservation purposes, and the current session is done logging the user out of the web page back to the log in screen.

2.4.2.22 CancelReservation

Name	Cancel Reservation
ID	CancelReservation
Version	1.0
Author	Osama Mansour
Date	9/6/13
Summary	User case for canceling a car reservation through an administrator from within the car rental system.
Basic Path	1. Upon the request of a customer to cancel a reservation. The administrator will be presented with a section within the administrator view called customers in which the administrator will click. 2. The system will provide a display for customer inventory with a search bar on top. 3. The administrator would search the customer by typing in the last name, first name order of the desired customer. The System scroll window will updates its result based on the name provided. 4. The administrator would then click on the desired customer. The system will provide a new page listing all the customers information along with a vehicle history button. 5. The administrator will click the rental history button**.** The administrator would click on the desired vehicle rental with an in process status. 6. The System will then produce a message asking does the administrator desire to cancel the following reservation in the process, and two buttons a Cancel reservation or exit button. 7. The administrator would click cancel reservation. The system would then update the status of the vehicle rented to an available status. Releasing the vehicle to other customers for rentals, and the customer rental history would update the changes on the scroll window display. 8. The administrator would then click the exit button at the bottom of the rental history window, and would be returned to the customers information page. The administrator would perform click another exit button to end up back to the administrator home view.
Alternative Paths	1. In step 6, if the administrator were to click exit the system would return the administrator to the customer car rental history page.
Exception Paths	1. In step 3, if the administrator searches a nonexistent customer the system will return a blank.

Extension Points	1. In step 7, If the administrator desired to another car reservation they would click on the next desired vehicle and click cancel reservation.
Triggers	A customer request to cancel a reservation through an administrator.
Assumption	The Customer has placed a 72 hour notice for the cancellation of a vehicle reservation.
Pre-conditions	The has received the request, and is logged into the car rental service system.
Post-conditions	The administrator has cancelled the following reservation(s) on the request of the customer. Releasing the Car back into the pool of rentals for that location.

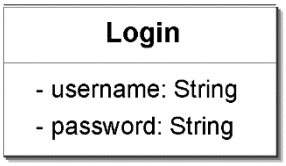
2.4.2.23 ReserveCar

Name	Reserve Car
ID	ReserveCar
Version	1.0
Author	Osama Mansour
Date	9/6/13
Summary	Use case for customers reserving a car through the car rental service.
Basic Path	1. The system will display a customer view of the home page with a list of options. One which will include a rent a car option. 2. The System will then display a new page with a scroll pane, and a list of available cars. The customer will click on his or her desired vehicle. 3. The system will display a new page with all the vehicle information and a button to rent a car. The customer would click on rent a car. 4. The system will update the database to store the car within the customers rental history, and remove the car for the pool of available cars, and return the user back to the customer home page. Charges will apply after the vehicle has been returned
Alternative Paths	1. In step 4, if the customer clicked cancel then he would be returned to the list of vehicles available.
Exception Paths	1. In step 2, if the system has no available cars it will display a screen with a message stating no cars are available. 2. In step 6, if the system has returned that the card was invalid. A message will appear stating that the credit card information was invalid.
Extension Points	1. In step 6, if vehicle is returned late then late fees will apply
Triggers	A customer wants to rent a vehicle.
Assumption	The Customer has not reached the max amount of rentals within one period
Pre-conditions	The customer is logged into the system and desires to rent a car.
Post-conditions	The customers has made his payment, and the car has been reserved for rental, and removed from the pool of available cars within the database.

2.4.3 Object Model

2.4.3.1 Login

Object



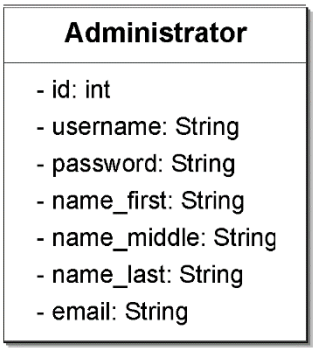
Description

username: unique ID for each user account in the system

password: string of characters used for user authentication

2.4.3.2 Administrator

Object



Description

id: unique user identification number

username: unique ID for each user account in the system

password: string of characters used for user authentication

name_first: person's given name

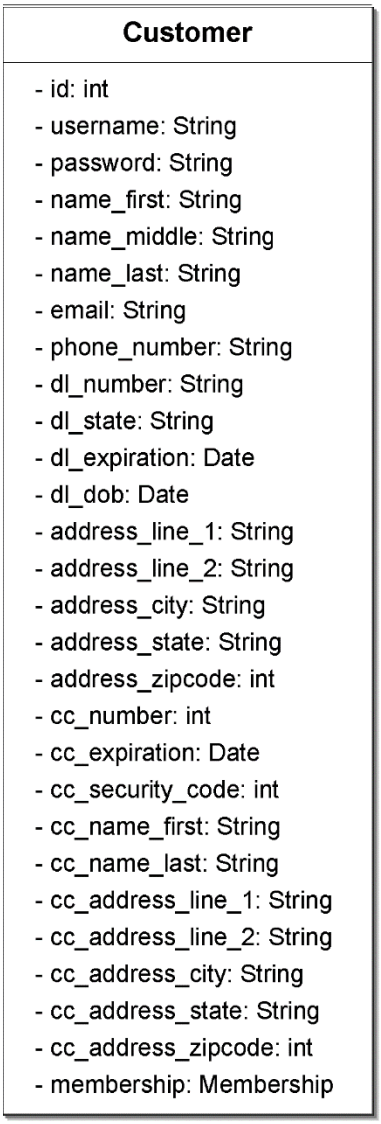
name_middle: person's middle name

name_last: person's family name

email: electronic mail address

2.4.3.4 Customer

Object

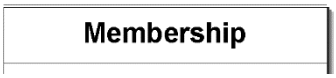


Description

id: unique user identification number
username: unique ID for each user account in the system
password: string of characters used for user authentication
name_first: person's given name
name_middle: person's middle name
name_last: person's family name
email: electronic mail address
phone_number: user telephone number
dl_number: driver license ID number
dl_state: driver license issued state
dl_expiration: driver license expiration date
dl_dob: driver license Date Of Birth (DOB)
address_line_1: address house number and street name
address_line_2: address apartment number or P.O. Box
address_city: address city
address_state: address state
address_zipcode: Zone Improvement Plan (ZIP) code
cc_number: credit card ID number
cc_expiration: credit card expiration date
cc_security_code: credit Card Security Code (CSC)
cc_name_first: credit card owner's given name
cc_name_last: credit card owner's family name
cc_address_line_1: credit card billing address house number and street name
cc_address_line_2: credit card billing address apartment number or P.O. Box
cc_address_city: credit card billing address city
cc_address_state: credit card billing address state
cc_address_zipcode: credit card billing address Zone Improvement Plan (ZIP) code
membership: object of Membership class

2.4.3.5 Membership

Object



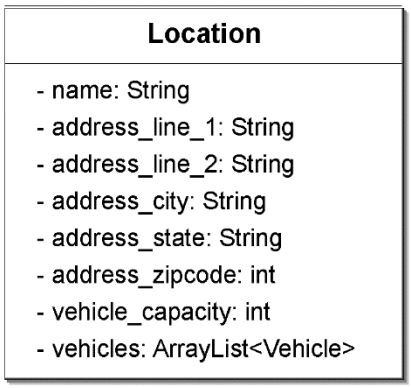
Description

name: unique title of a membership tier

monthly_price: cost in USD for one (1) month

2.4.3.6 Location

Object



Description

name: unique title of a rental location

address_line_1: address house number and street name

address_line_2: address apartment number or P.O. Box

address_city: address city

address_state: address state

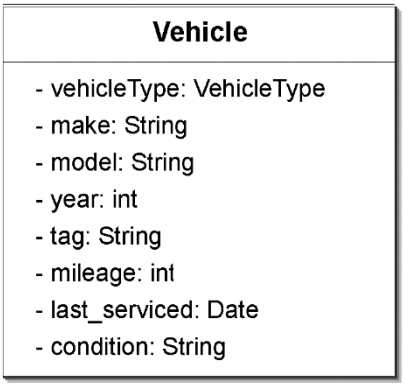
address_zipcode: address Zone Improvement Plan (ZIP) code

vehicle_capacity: number of vehicles the location can accommodate

vehicles: list of Vehicle class objects

2.4.3.7 Vehicle

Object



Description

vehicleType: object of VehicleType class

make: name of the automobile manufacturer

model: particular brand of vehicle sold under a marque by a manufacturer

year: model year of the vehicle

tag: unique numeric or alphanumeric code for the vehicle registration plate

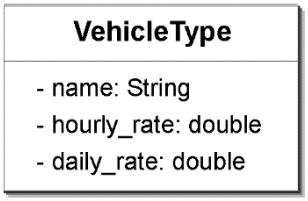
mileage: vehicle miles traveled

last_serviced: last time vehicle was serviced in date format

condition: vehicle condition specified as: good, needs cleaning, needs maintenance, etc.

2.4.3.8 VehicleType

Object



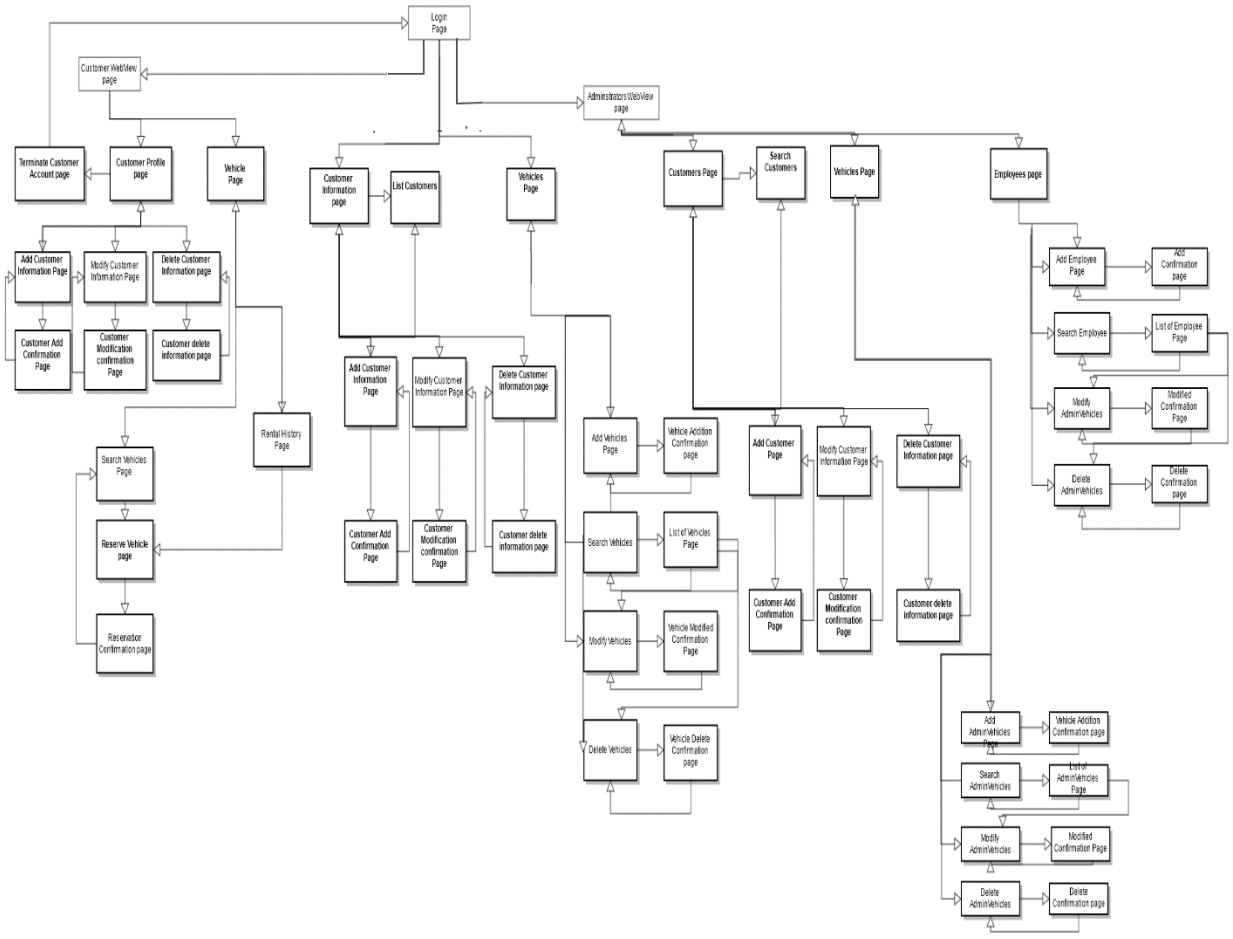
Description

name: unique title of a vehicle type

hourly_rate: cost in USD for one (1) hour

daily_rate: cost in USD for one (1) day

2.4.4 Navigation path and User Interface



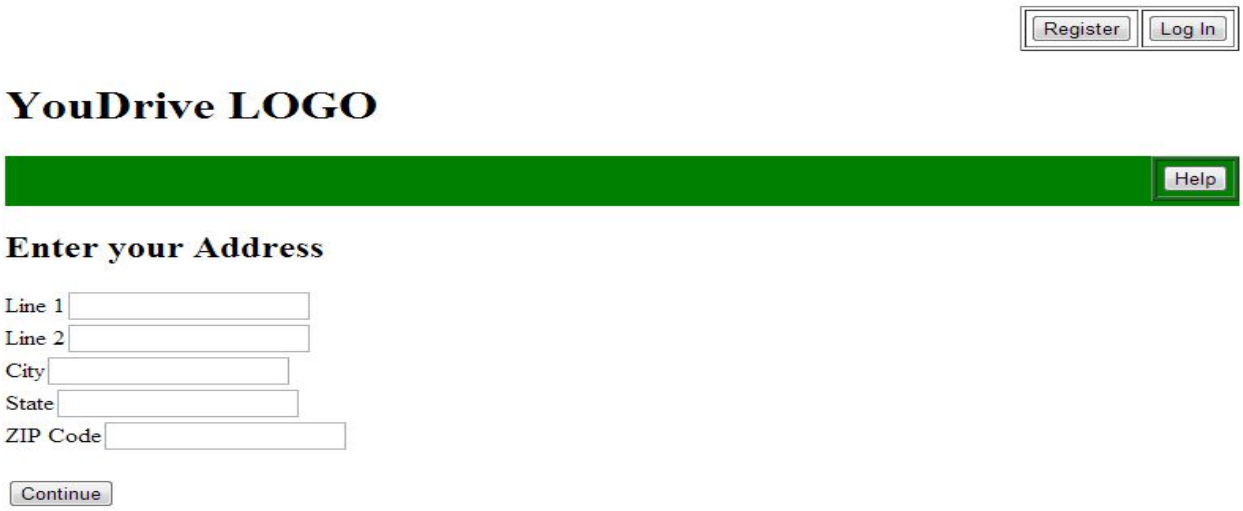
2.4.4.1 Navigation path

2.4.4.2 User Interface Mock-Up

The following user Interface shoots are a mock-up of the basic standards of how the system begin to be developed on a front end level. We have presented a sequential user interface mock up 1-14 illustrating an idea of the basic systems functions displaying how they will feel and operate.

2.4.4.3 Registration of User Profile

2.4.4.4 Registration of User Address Information



2.4.4.5 Choose of Driving Plans

Register | Log In

YouDrive LOGO

Help

Pick a Driving Plan

1-Month	6-Month
Occational Driving Plan	Extra Value
$25 one-time activation fee	$20 one-time activation fee
$10/month	$8/month
Select this Plan	Select this Plan

2.4.4.6 Registration of Payment Information

YouDrive LOGO

Help

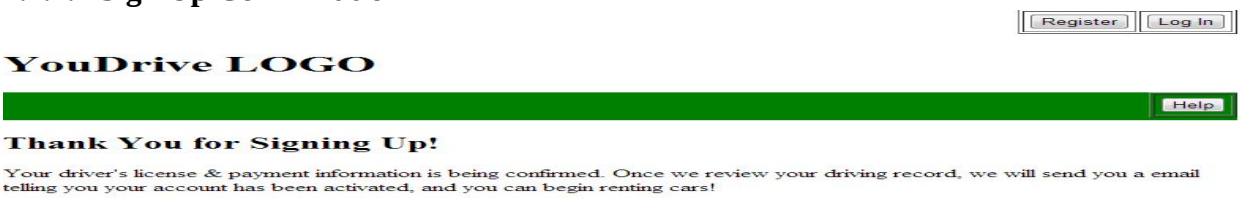
Enter your Payment Info

Credit Card #
Security Code
Expires
September 1987
Cardholder First Name
Cardholder Last Name
License Issued in Alabama
Driver's License #
Re-type License #

Billing Address
Line 1
Line 2
City
State
ZIP Code

Continue

2.4.4.7 Sign Up Confirmation

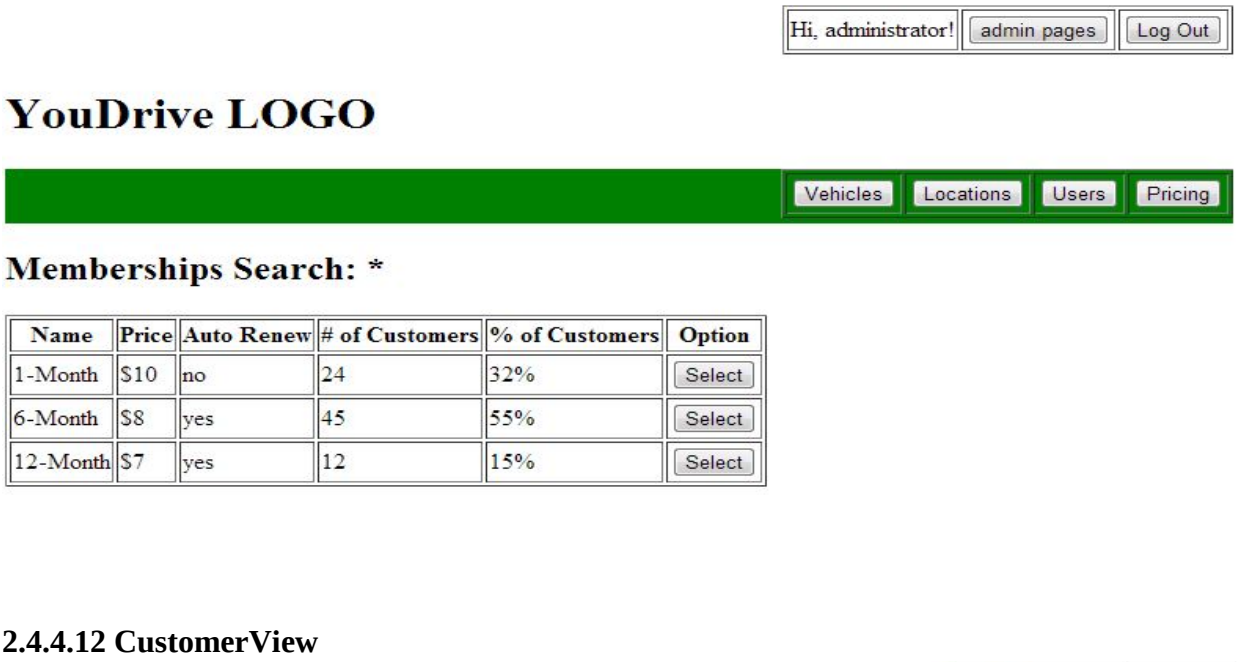


2.4.4.8 AddUser

2.4.4.9 AddVehicles

2.4.4.10 Search Vehicle By Location

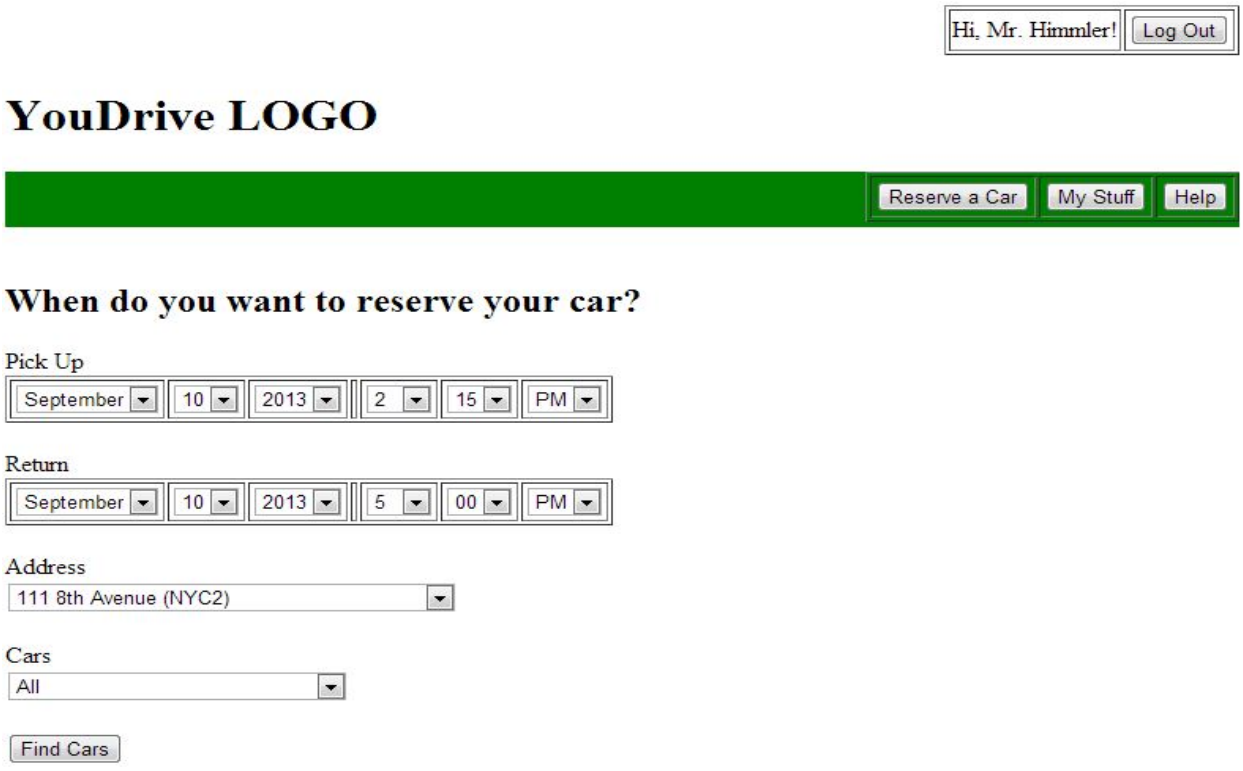
2.4.4.11 Admin Membership Search



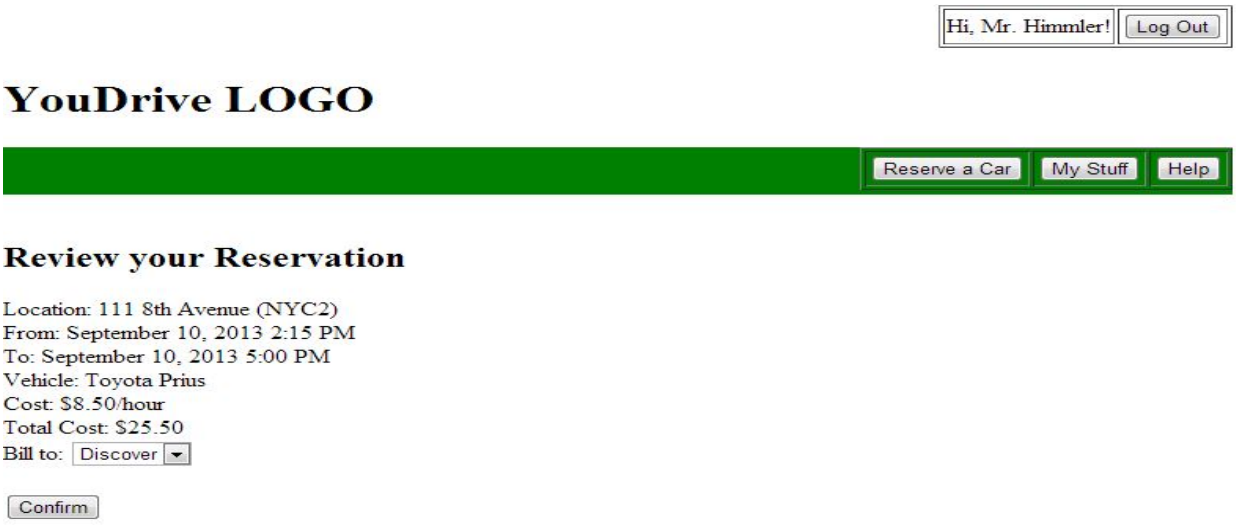
2.4.4.12 CustomerView

2.4.4.13 Vehicle Search

2.4.4.14 Car Reservation



2.4.4.15 Car Reservation Information



2.4.4.16 Reservation Confirmation


Hi, Mr. Himmler!
Log Out
YouDrive LOGO
Reserve a Car
My Stuff
Help
Success
Your vehicle has been reserved. A email confirmation will be sent out to confirm your reservation. Further instructions will be listed.

Analysis Document:
Car Rental Service

Group 4
CSCI 4050: Software Engineering
Instructor: Krys Kochut
Osama Mansour, Stephen Patton, Vincent Lee, and Minh Pham

Table of Contents

2.1.1 Object Model

2.1.1.1 Data Dictionary

2.1.1.1.1 Entity Package

Class Name	User		
Purpose	The generalize entity involved in a Car rental system transaction		
SuperClass	None		
SubClass	Administrator, Customer		
Attributes	Name	Type	Description
	UserID	String	The ID of the User as defined by the System Registration
	password	String	The encryption on the user profile to be accessed
	role	String	The role of the user defined by the system
Operations	Name	Inputs	Description
	Login	UserID, Password role	The login page allows users access to registered users within a system
	Logout		The logout page allows users currently login in to exit there current system profile
	updateProfile		The update profile operation allows for user to change and save their profile information
Relationships	Super class of Administrator, Customer		

Class Name	DriversLicense		
Purpose	Provide driver's license information of Customer		
SuperClass	None		
SubClass	None		
Attributes	Name	Type	Description
	number	String	The number on the license
	expiration	String	The expiration date
	dateOfBirth	String	The date of birth on the license
Operations	Name	Inputs	Description
Relationships	None		

Class Name	CreditCard		
Purpose	Provide credit card information of Customer		
SuperClass	None		
SubClass	None		
Attributes	Name	Type	Description
	number	String	The number on the card
	expiration	String	The expiration date
	code	Int	The CVC code
Operations	Name	Inputs	Description
Relationships	None		

Class Name	Customer			
Purpose	The generalization of group of Users labeled Customers			
SuperClass	Users			
SubClass	None			
Attributes	Name	Type	Description	
	Social Security Number	String	Customer social security information for verification	
	firstName	String	Customer First name	
	lastName	String	Customer Last name	
	address	String	Customer Address	
	telephone	String	Customer Telephone number	
Operations	Name	Inputs	Outputs	Description
	ViewMyInfo		Personal Information Display	Displays all personal information of the current logged in Customer
	ViewAvailbileCars	CarName CarLocation	Available Cars	Displays the Cars available for rental by location
	ViewMyRentalHistory		Rental History Report	Displays a current logged Customer rental history
	ViewMyBillingInfo		Billing Report	Displays Current logged Customer billing report
	ViewMyCarRental		Car Rental Report	Displays Customer past and present car rentals
	EditMyInfo	FirstName LastName Address Telephone #	Updated Personal Information	Allows Customers to edit their information as needed

Relationships	Subclass of the User A Customer can only rent 1 Car at a time A Customer can only edit personal information, but not vehicle information

Class Name	Administrator			
Purpose	The generalization of group of Users labeled Administrator within the Car rental System			
SuperClass	Users			
SubClass	None			
Attributes	Name	Type	Description	
	AdminID	Int	ID identifying the Administrator within the car rental system	
	firstName	String	First name	
	lastName	String	Last name	
	address	String	Address	
	telephone	String	Telephone number	
Operations	Name	Inputs	Outputs	Description
	ViewMyInfo		Personal Information Display	Displays all personal information of the current logged in administrator
	ViewVehicles	CarName CarLocation	Available Cars including model, make year, vin number, and repair history reports.	Displays the Cars available within the Car rental system
	ViewRentalHistory	Customer Name	Rental History Report on any Customer within the system	Displays a rental history for all customer by name
	ViewBillingHistory	Customer Name	Billing Report	Displays all Customer History Billing reports
	ViewCarRentals	Customer Name	Car Rental Report	Displays All Customer Car rental reports

	EditVehicleInfo	Vehicle Name	Updated Vehicle Information	Edit Vehicle Information
	EditCustomerInfo	CustomerName	Update Customer Personal Information	Allows Admin to edit customer information both adding and removing Customers
Relationships	Subclass of User An administrator can edit Vehicle Information, and Customer information on Request			

Class Name	VehicleType			
Purpose	An entity class for storing information on vehicle types			
SuperClass	None			
SubClass	None			
Attributes	Name	Type	Description	
	name	String	The name of the vehicle type	
	hourly_rate	Double	The hourly price of the vehicle type	
	daily_rate	Double	The daily price of the vehicle type	
Operations	Name	Inputs	Outputs	Description
Relationships	None			

Class Name	Vehicle			
Purpose	An entity class for storing information on vehicles in the fleet			
SuperClass	None			
SubClass	None			
Attributes	Name	Type	Description	
	make	String	The make of the vehicle	
	model	String	The model of the vehicle	
	year	Integer	The manufacture year of the vehicle	
	tag	String	The registration tag of the vehicle	
	mileage	Integer	The current mileage of the vehicle	
	last_serviced	Date	The last date the vehicle was serviced	
	condition	String	The condition of the vehicle	
Operations	Name	Inputs	Outputs	Description
Relationships	A vehicle belongs to 1 location			

Class Name	Location			
Purpose	An entity class for storing information on point of presence POP locations			
SuperClass	None			
SubClass	None			
Attributes	Name	Type	Description	
	name	String	The name of the location	
	address	String	The address of the location	
	vehicle_capacity	Integer	The capacity of the location	
	vehicles	ArrayList<Vehicle>	The list of vehicles assigned to the location	
Operations	Name	Inputs	Outputs	Description
Relationships	None			

Class Name	Membership			
Purpose	An entity class for storing information on membership tiers			
SuperClass	None			
SubClass	None			
Attributes	Name	Type	Description	
	name	String	The name of the membership tier	
	monthly_price	Double	The price of one month's service	
	duration	Integer	The length of membership in months	
Operations	Name	Inputs	Outputs	Description
Relationships	A customer belongs to 1 membership			

2.1.1.1.2 user.control package

Class Name	User			
Purpose	This control class allows a User to complete actions associated with login and profile modification			
SuperClass	None			
SubClass	None			
Attributes	Name	Type	Description	
Operations	Name	Inputs	Outputs	Description

	login	username password		Logs the user in with the appropriate permissions
	logout			Logs the user out
	loadProfile	username		Gets all relevant information related to the username
	updateProfile	profile	Boolean	Updates profile information and returns confirmation with Boolean
	registerCustomer	username password address creditcard driverlicense	Boolean	Allows a User to register as a customer
	vehicleSearch	name		Performs a search of vehicles in the fleet
Relationships	None			

2.1.1.1.3 user.boundary package

Class Name	LoginUI		
Purpose	This boundary class allows a User to log in		
SuperClass	None		
SubClass	None		
Attributes	Name	Type	Description

Operations	Name	Inputs	Outputs	Description
	login	username password		Verify the user and logs the user with appropriate permissions
Relationships	None			

Class Name	LogoutUI			
Purpose	This boundary class allows a User to log out			
SuperClass	None			
SubClass	None			
Attributes	Name	Type	Description	
Operations	Name	Inputs	Outputs	Description
	logout			Logs the user out
Relationships	None			

Class Name	ChangeProfileUI			
Purpose	This boundary class allows a User to perform profile changes			
SuperClass	None			
SubClass	None			
Attributes	Name	Type	Description	
Operations	Name	Inputs	Outputs	Description
	updateProfile	profile		Updates the profile of a user
	saveProfile	profile	Boolean	Saves changes of profile change to data structure
Relationships	None			

Class Name	RegisterUI
Purpose	This boundary class allows a User to register as a customer
SuperClass	None
SubClass	None

Attributes	Name	Type	Description	
Operations	Name	Inputs	Outputs	Description
	registerCustomer	username password address creditcard driverlicense	Boolean	Registers a customer
Relationships	None			

Class Name	SearchUI			
Purpose	This boundary class allows a User to search the vehicle fleet			
SuperClass	None			
SubClass	None			
Attributes	Name	Type	Description	
Operations	Name	Inputs	Outputs	Description
	serach	keyword	List	Gets a list of available vehicles based on keyword
Relationships	None			

2.1.1.1.4 administrator.control package

Class Name	UserAdd			
Purpose	This control class allows Administrator to add users of all permissions			
SuperClass	None			
SubClass	None			
Attributes	Name	Type	Description	
Operations	Name	Inputs	Outputs	Description
	addUser	username password	Boolean	Adds a user to the database with Boolean confirmation
Relationships	None			

Class Name	VehicleRemove			
Purpose	This control class allows Administrator to remove a vehicle from the system			
SuperClass	None			
SubClass	None			
Attributes	Name	Type	Description	
Operations	Name	Inputs	Outputs	Description
	removeVehicle	vehicleProfile	Boolean	Removes a vehicle from the fleet with Boolean conformation
Relationships	None			

Class Name	VehicleTypeRemove			
Purpose	This control class allows a Administrator to remove a vehicle type from the system			
SuperClass	None			
SubClass	None			
Attributes	Name	Type	Description	
Operations	Name	Inputs	Outputs	Description
	removeVehicleType	vehicleType	Boolean	Removes a vehicletype from the database with confirmation
Relationships	None			

2.1.1.1.5 administrator.boundary package

Class Name	UserAddUI			
Purpose	This boundary class allows an Administrator to add a user to the domain			
SuperClass	None			
SubClass	None			
Attributes	Name	Type	Description	
Operations	Name	Inputs	Outputs	Description
	addUser	username password	Boolean	Adds a user to the system confirmation

Relationships	None

Class Name	RemoveVehicleUI			
Purpose	This boundary class allows Administrator to remove a vehicle from the fleet			
SuperClass	None			
SubClass	None			
Attributes	Name	Type	Description	
Operations	Name	Inputs	Outputs	Description
	removeVehicle	vehicleProfile	Boolean	Removes a vehicle from the fleet with conformation
Relationships	None			

Class Name	RemoveVehicleTypeUI			
Purpose	This boundary class allows Administrator to remove a vehicletype from the system			
SuperClass	None			
SubClass	None			
Attributes	Name	Type	Description	
Operations	Name	Inputs	Outputs	Description
	removeVehicleType	vehicleType	Boolean	Removes a vehicletype from the site with confirmation
Relationships	None			

Class Name	VehicleTypeUI			
Purpose	This boundary class allows a Administrator to perform actions on the vehicle types			
SuperClass	None			
SubClass	None			
Attributes	Name	Type	Description	
Operations	Name	Inputs	Outputs	Description
	addVehicleType	name hourly_rate daily_rate	Boolean	Adds a vehicle type to the system with Boolean confirmation
	modifyVehicleType	vehicleType	Boolean	Modifies a vehicle type information with Boolean confirmation
Relationships	None			
Class Name	UserUI			
Purpose	This boundary class allows an Administrator to view information on users			
SuperClass	None			
SubClass	None			
Attributes	Name	Type	Description	
Operations	Name	Inputs	Outputs	Description
	listUser	keyword	List	Lists users based on keyword match
Relationships	None			

2.1.1.1.8 customer.control package

Class Name	Membership			
Purpose	This control class allows a Customer to perform actions to their membership options			
SuperClass	None			
SubClass	None			
Attributes	Name	Type	Description	
Operations	Name	Inputs	Outputs	Description
	terminateMembership		Boolean	Terminates a membership of a customer with Boolean confirmation
Relationships	None			

Class Name	VehicleReservation			
Purpose	This control class allows a Customer to preform actions related to renting a vehicle			
SuperClass	None			
SubClass	None			
Attributes	Name	Type	Description	
Operations	Name	Inputs	Outputs	Description
	reserveVehicle	vehicleProfile	Boolean	Reserves a vehicle with Boolean confirmation
	returnVehicle	vehicleProfile	Boolean	Checks a vehicle back into the fleet of available vehicles with Boolean confirmation
Relationships	None			

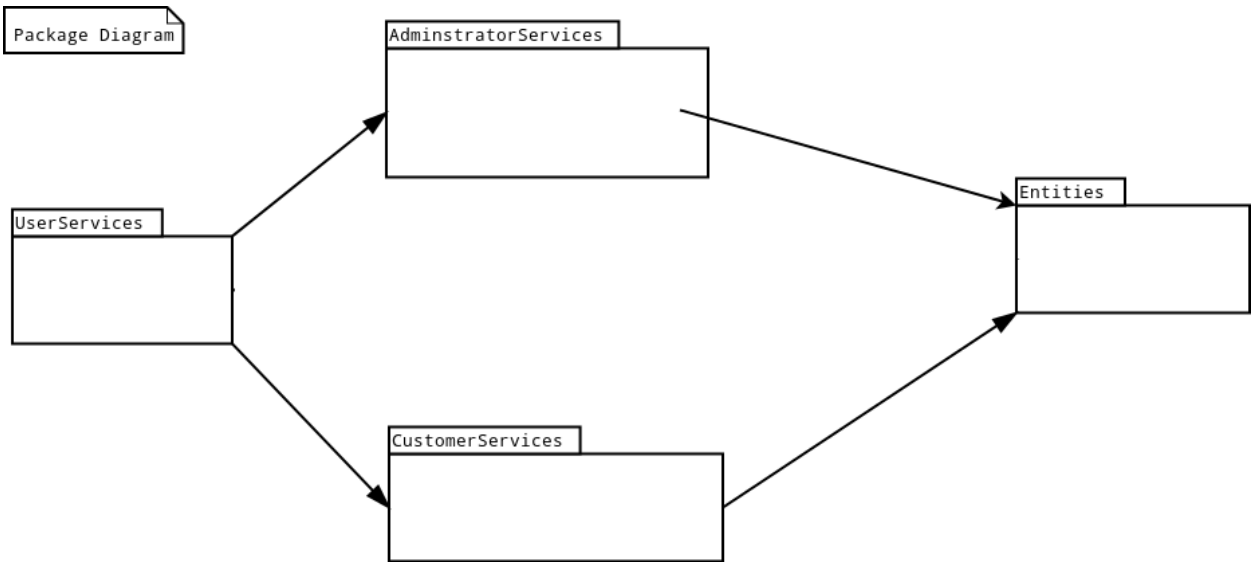
2.1.1.1.9 customer.boundary package

Class Name	MembershipUI			
Purpose	This boundary class allows a Customer to terminate their membership			
SuperClass	None			
SubClass	None			
Attributes	Name	Type	Description	
Operations	Name	Inputs	Outputs	Description
	terminateMembership		Boolean	Terminates a customer's membership with conformation
Relationships	None			

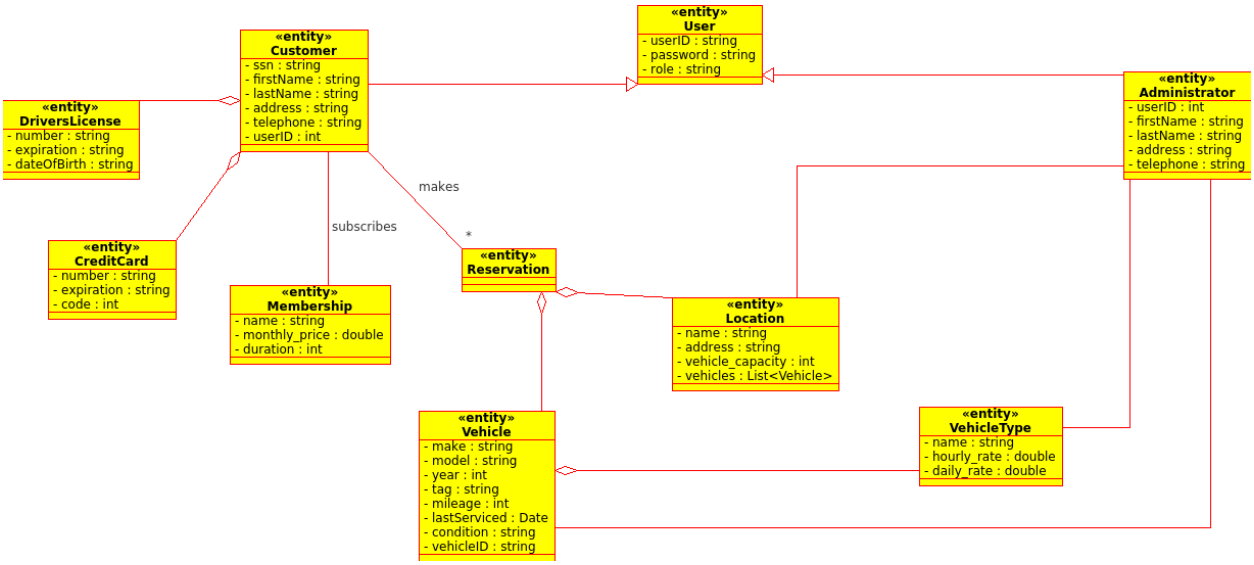
Class Name	ReservationUI			
Purpose	This boundary class allows a Customer to reserve a vehicle			
SuperClass	None			
SubClass	None			
Attributes	Name	Type	Description	
Operations	Name	Inputs	Outputs	Description
	reserveVehicle	vehicleProfile	Boolean	Reserves a vehicle with Boolean confirmation
	returnVehicle	vehicleProfile	Boolean	Checks a vehicle back into the fleet of available vehicles with Boolean confirmation
Relationships	None			

2.1.1.2 Class Diagrams

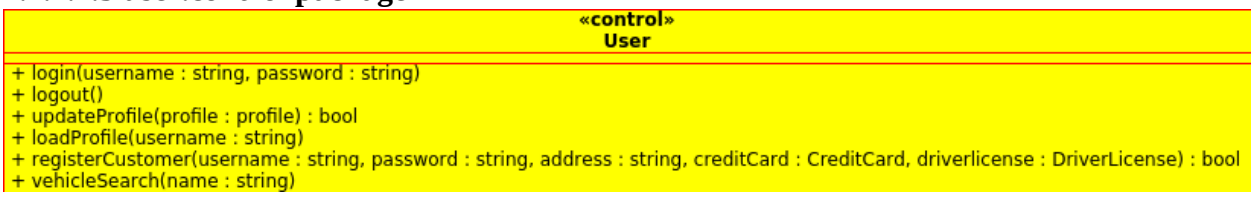
2.1.1.2.1 Package Diagram



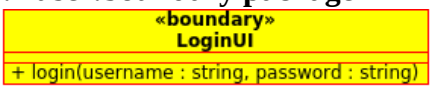
2.1.1.2.2 Entity Package

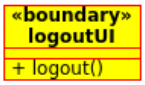


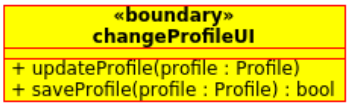
2.1.1.2.3 user.control package

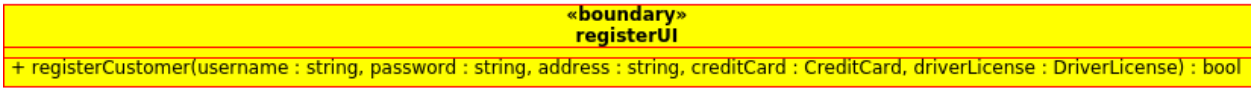


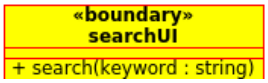
2.1.1.2.4 user.boundary package



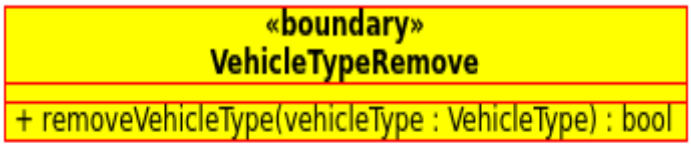




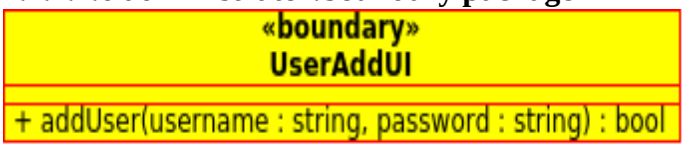


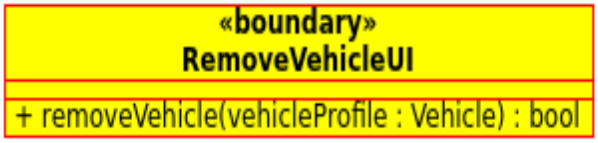


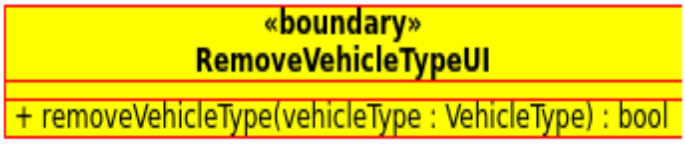
2.1.1.2.5 administrator.control package



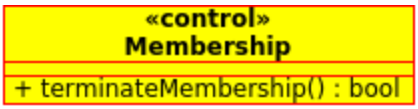
2.1.1.2.6 administrator.boundary package



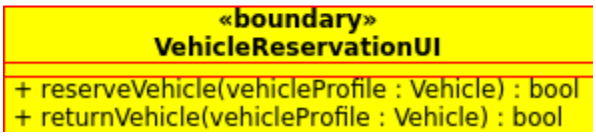




2.1.1.2.9 customer.control package

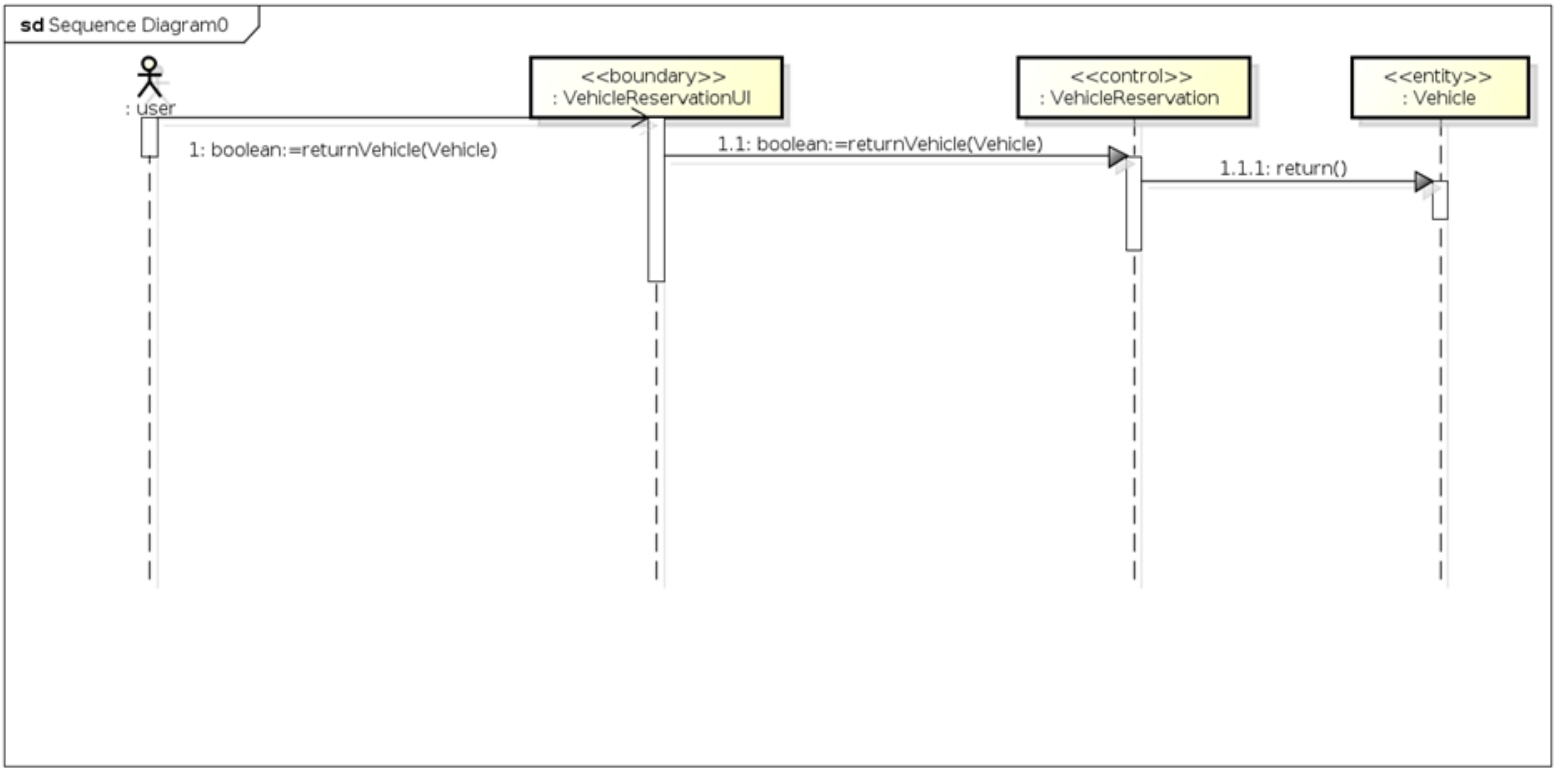


2.1.1.2.10 customer.boundary package

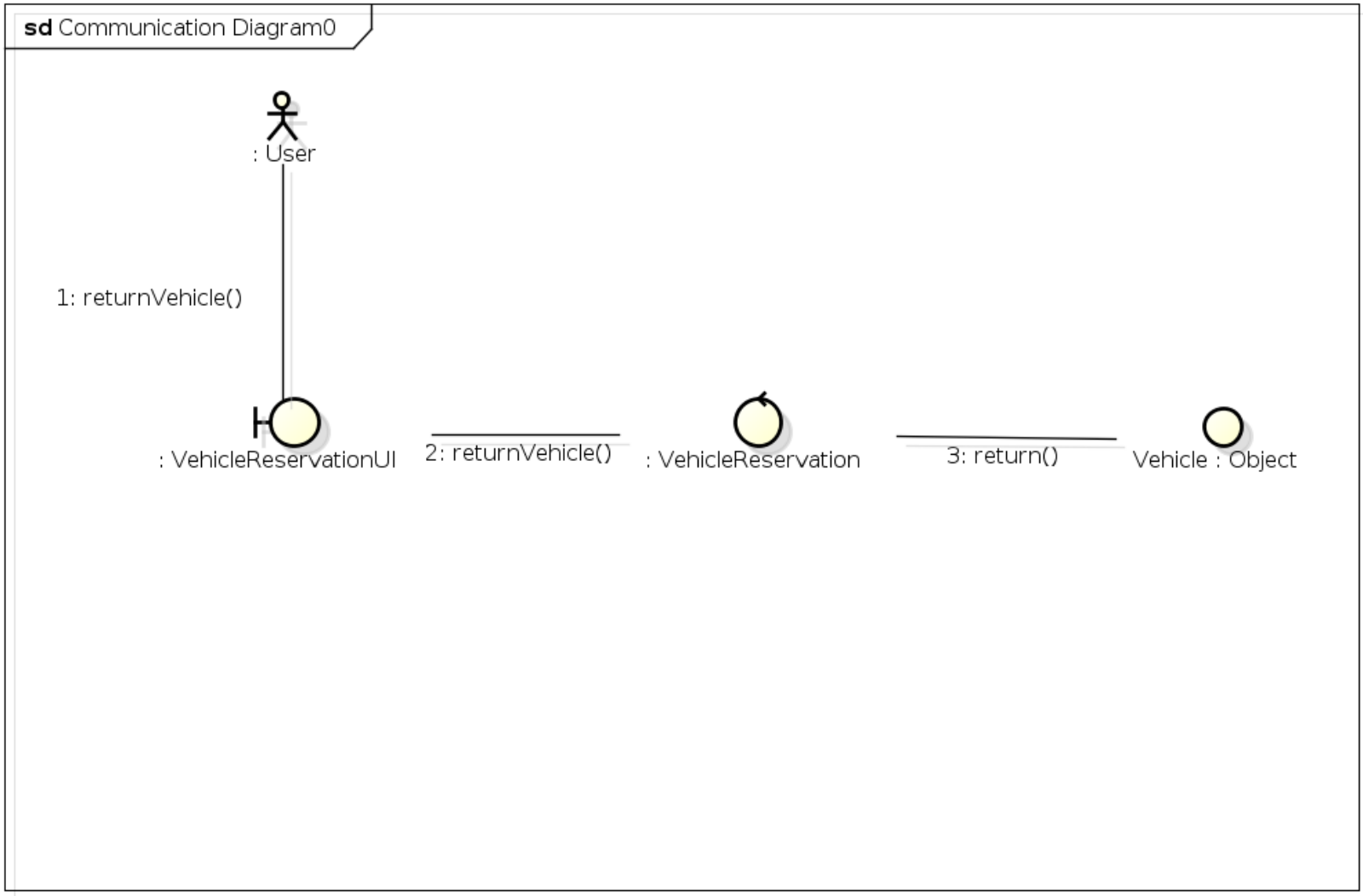


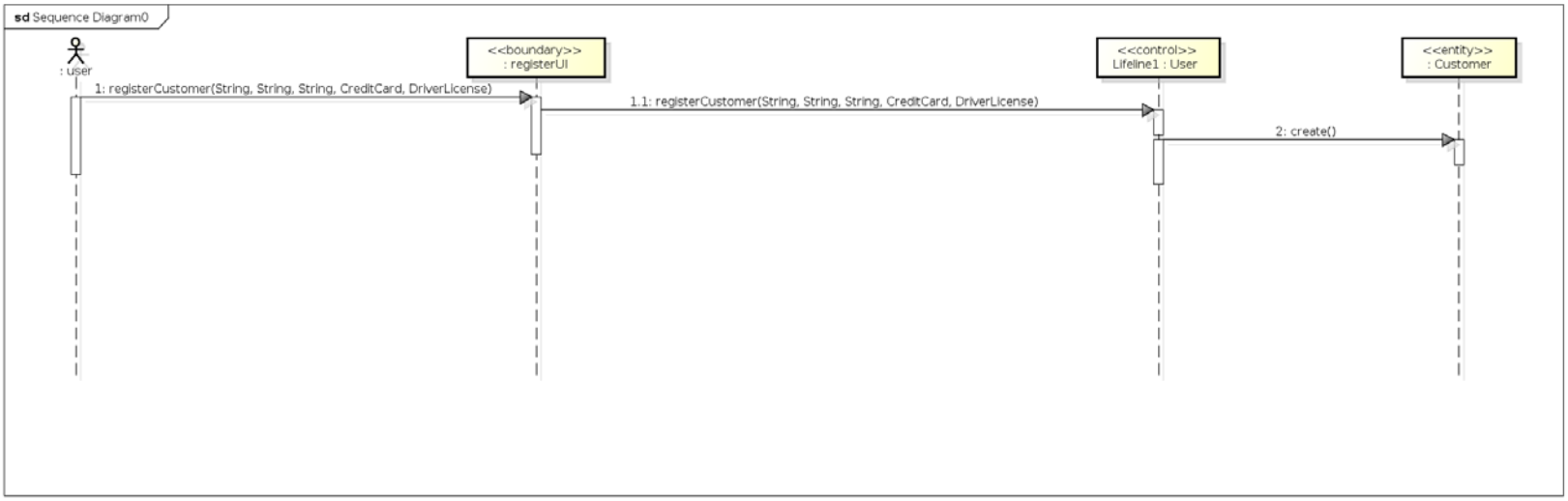
2.1.2 Dynamic Model – Sequence & Communication Diagrams

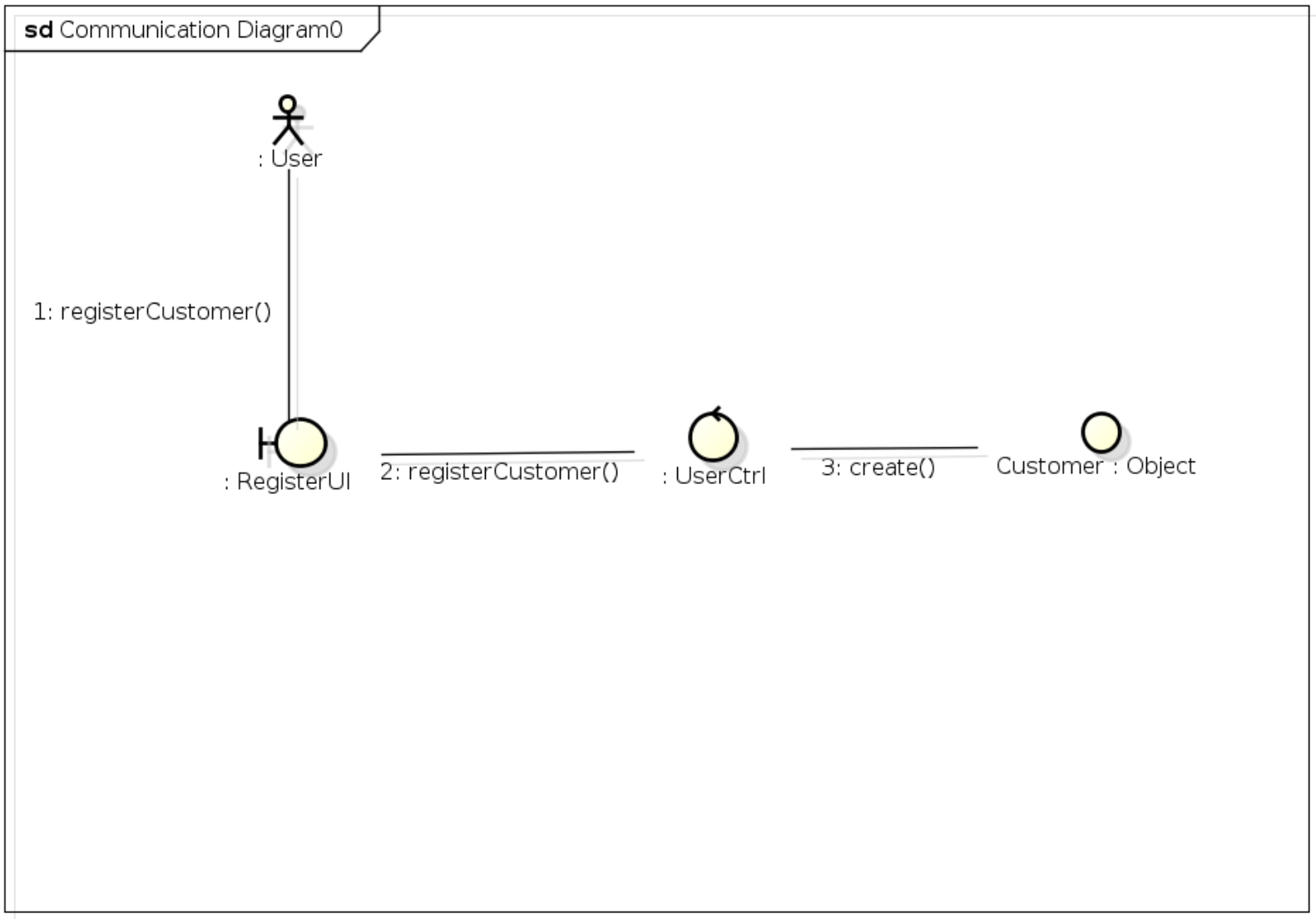
2.1.2.1 ReturnVehicle – Sequence Diagram



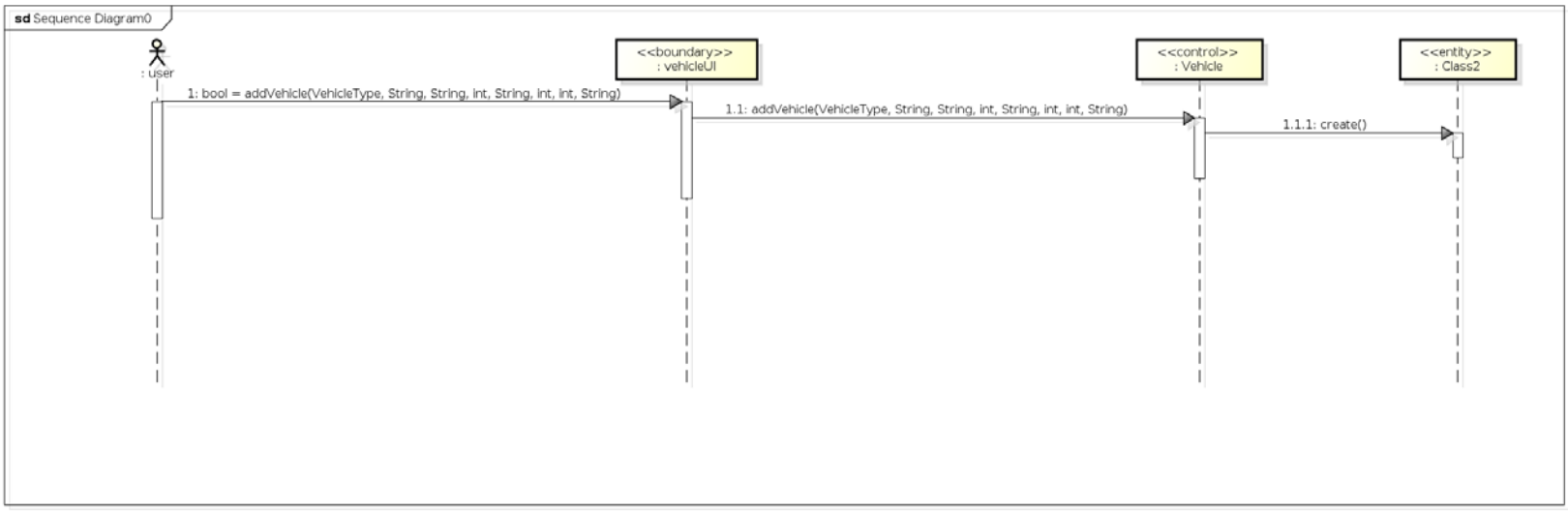
2.1.2.1 ReturnVehicle – Communication Diagram



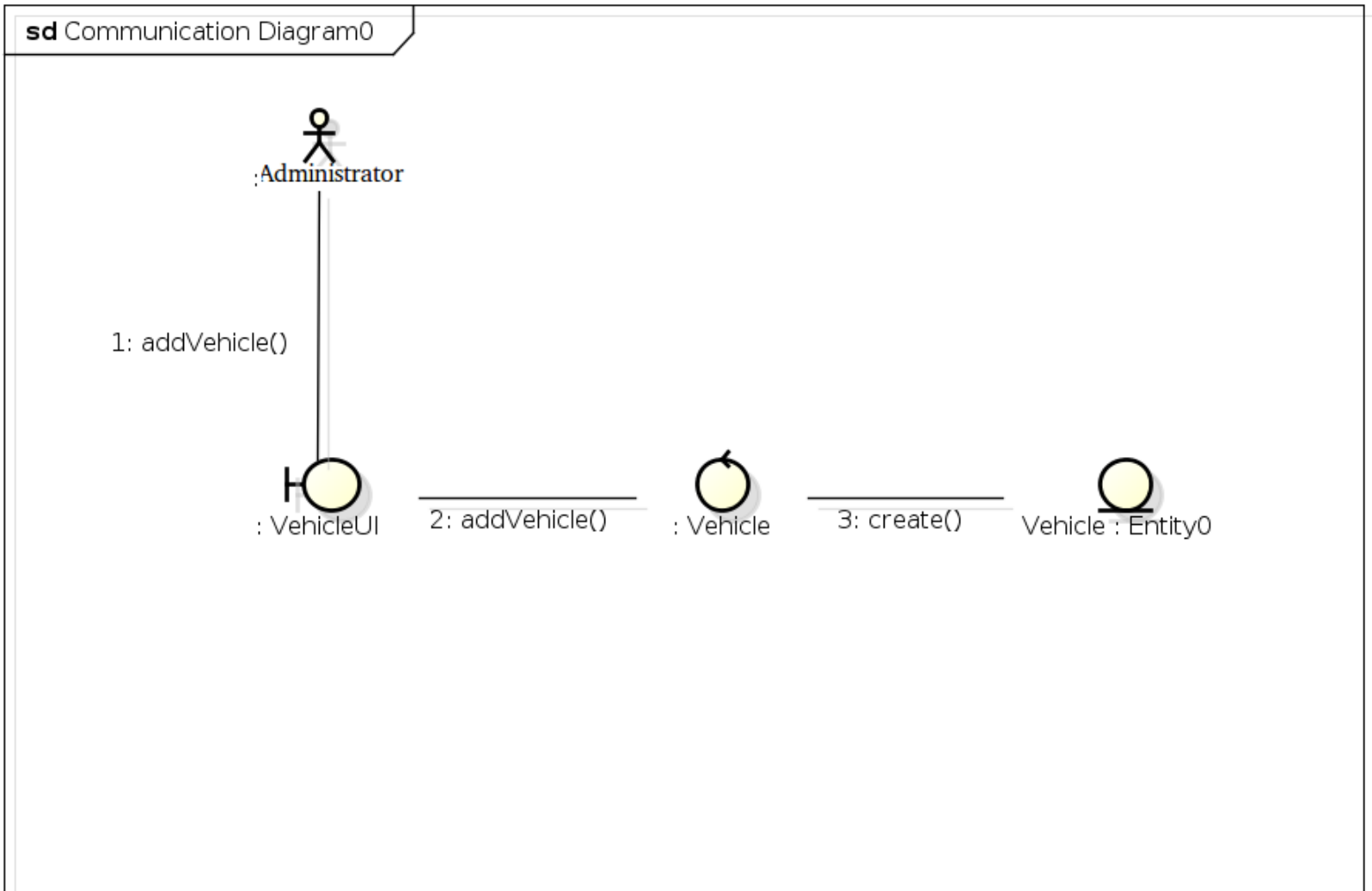



sd Communication Diagram0
: User
1: registerCustomer()
: RegisterUI
2: registerCustomer()
: UserCtrl
3: create()
Customer : Object

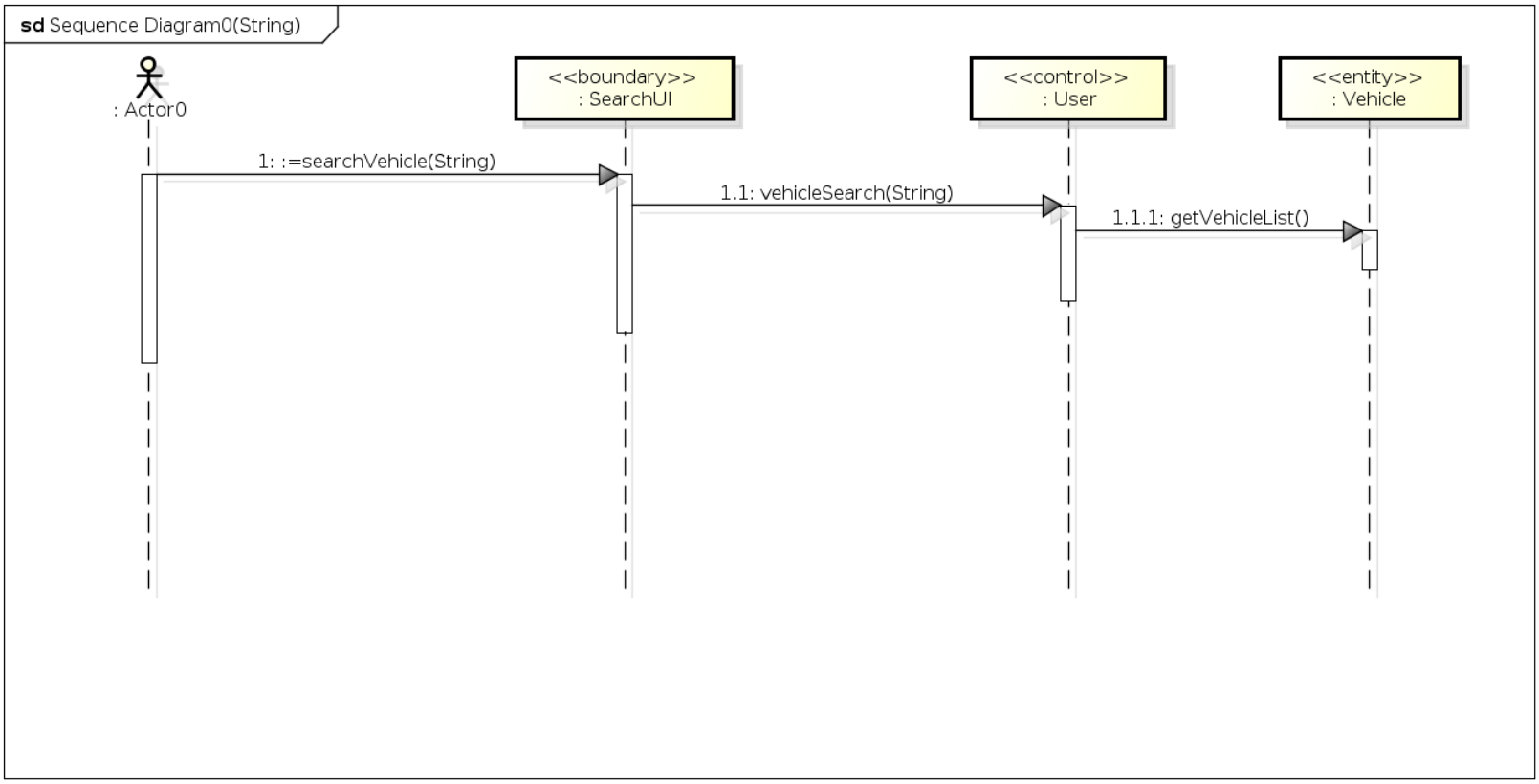
2.1.2.3 AddVehicle – Sequence Diagram



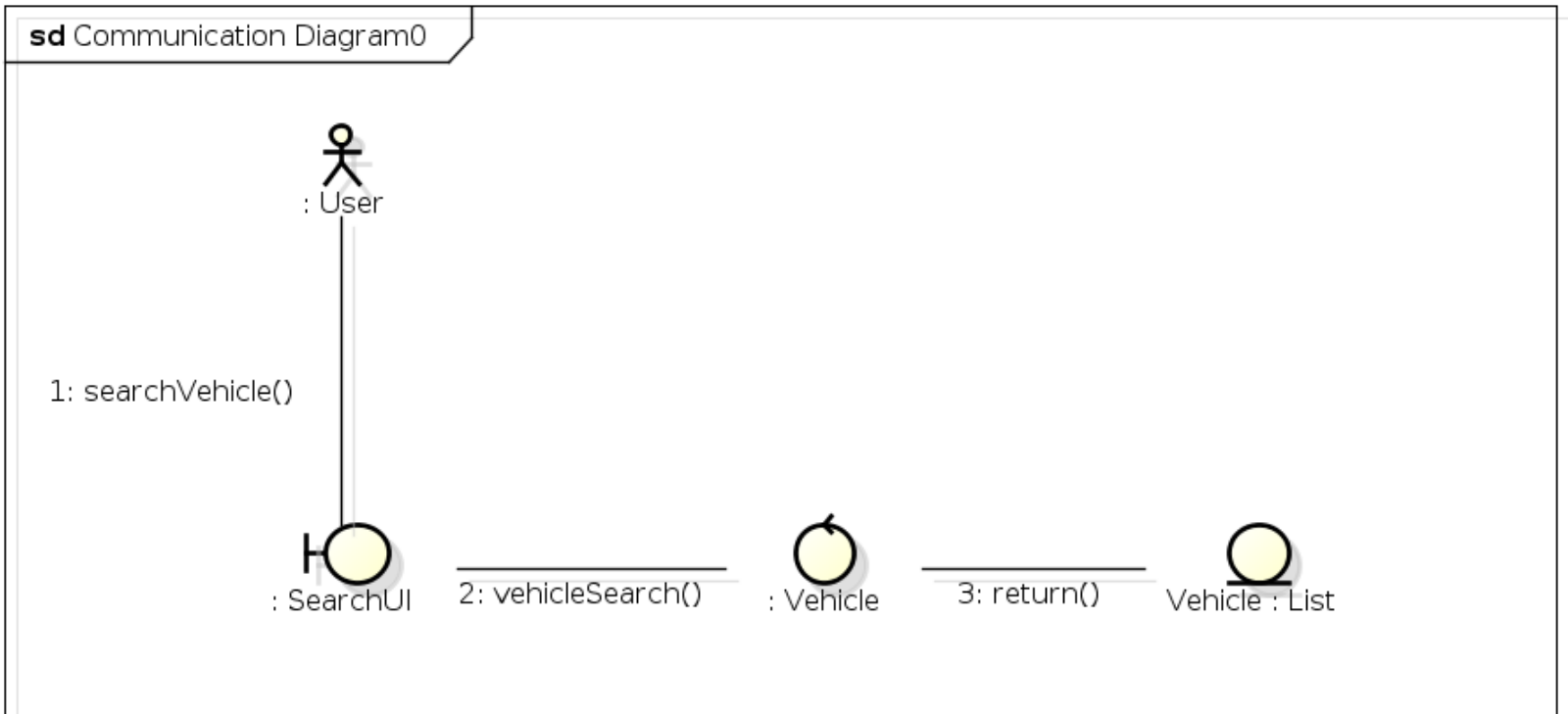
2.1.2.3 AddVehicle – Communication Diagram



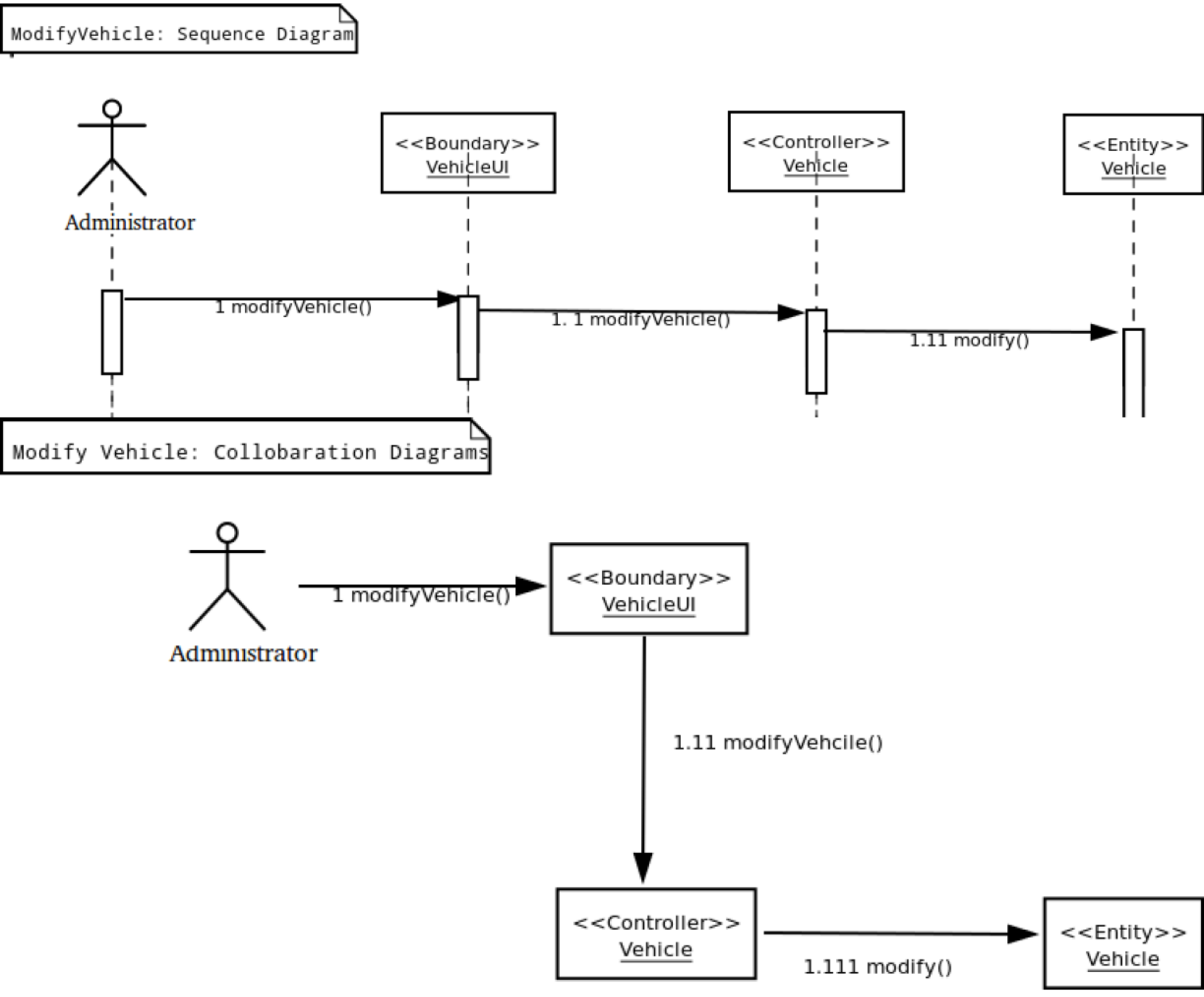
2.1.2.4 ListVehicle – Sequence Diagram



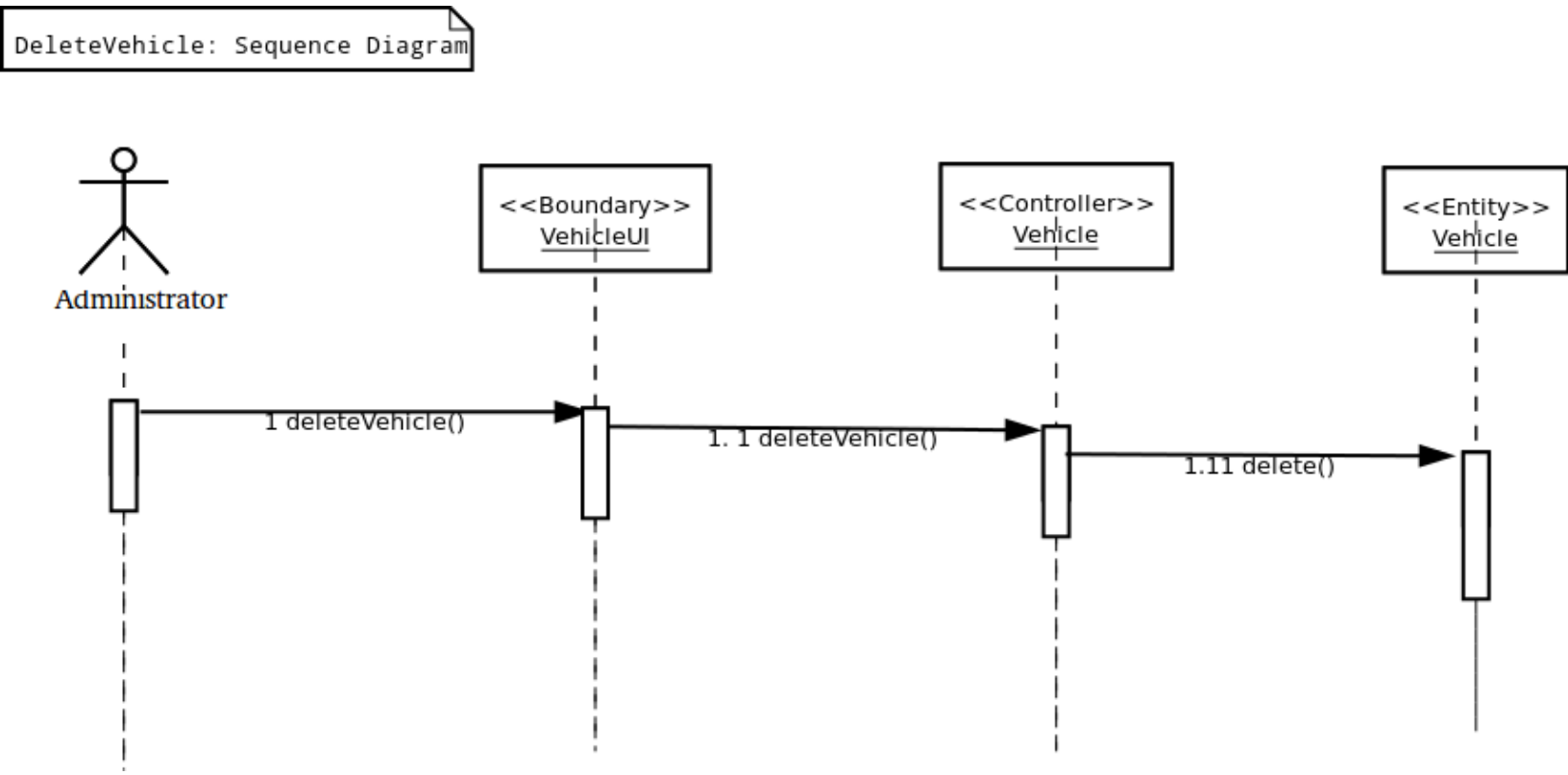
2.1.2.4 ListVehicle – Communication Diagram



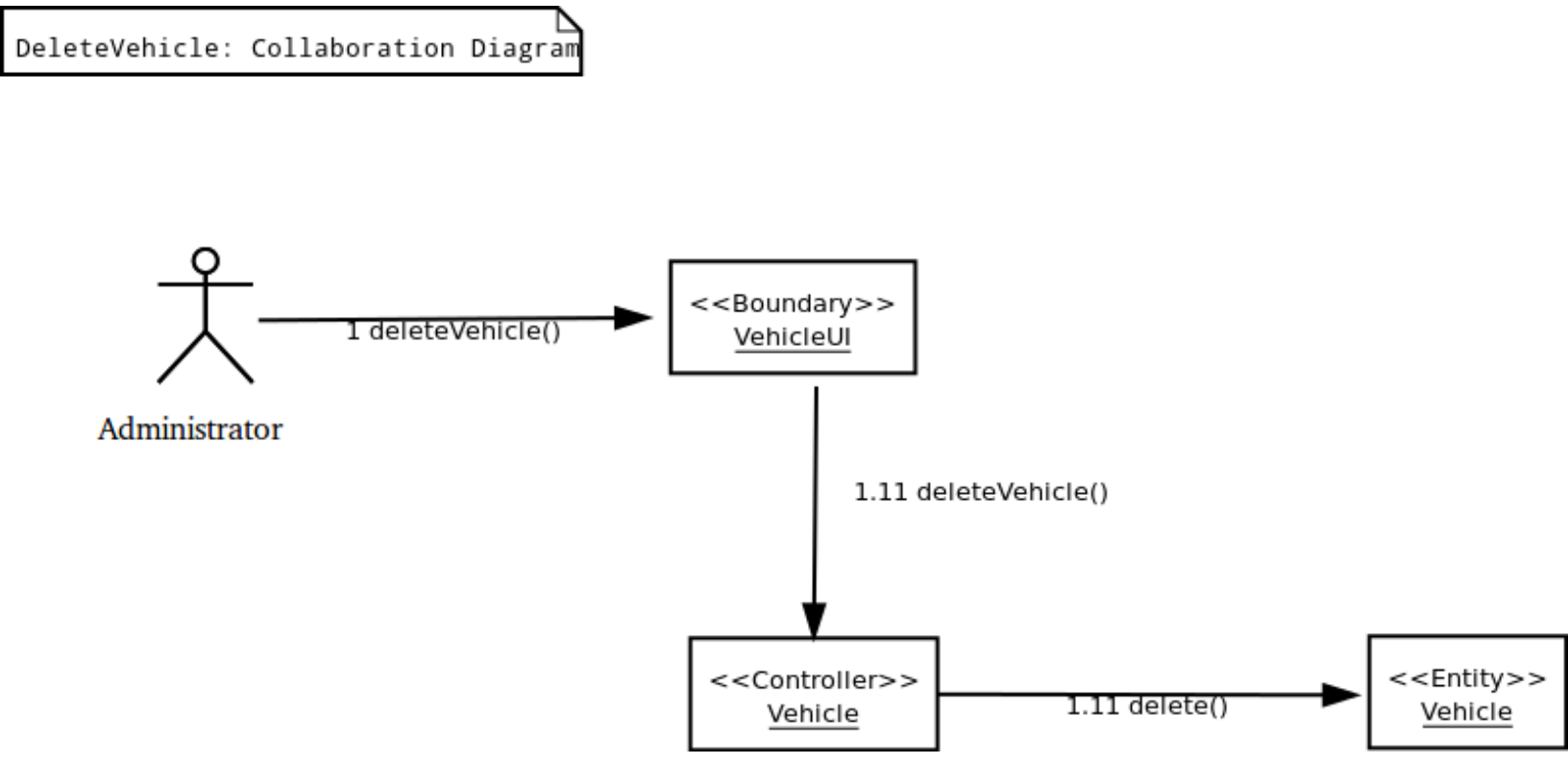
2.1.2.5 ModifyVehicle – Sequence Diagram



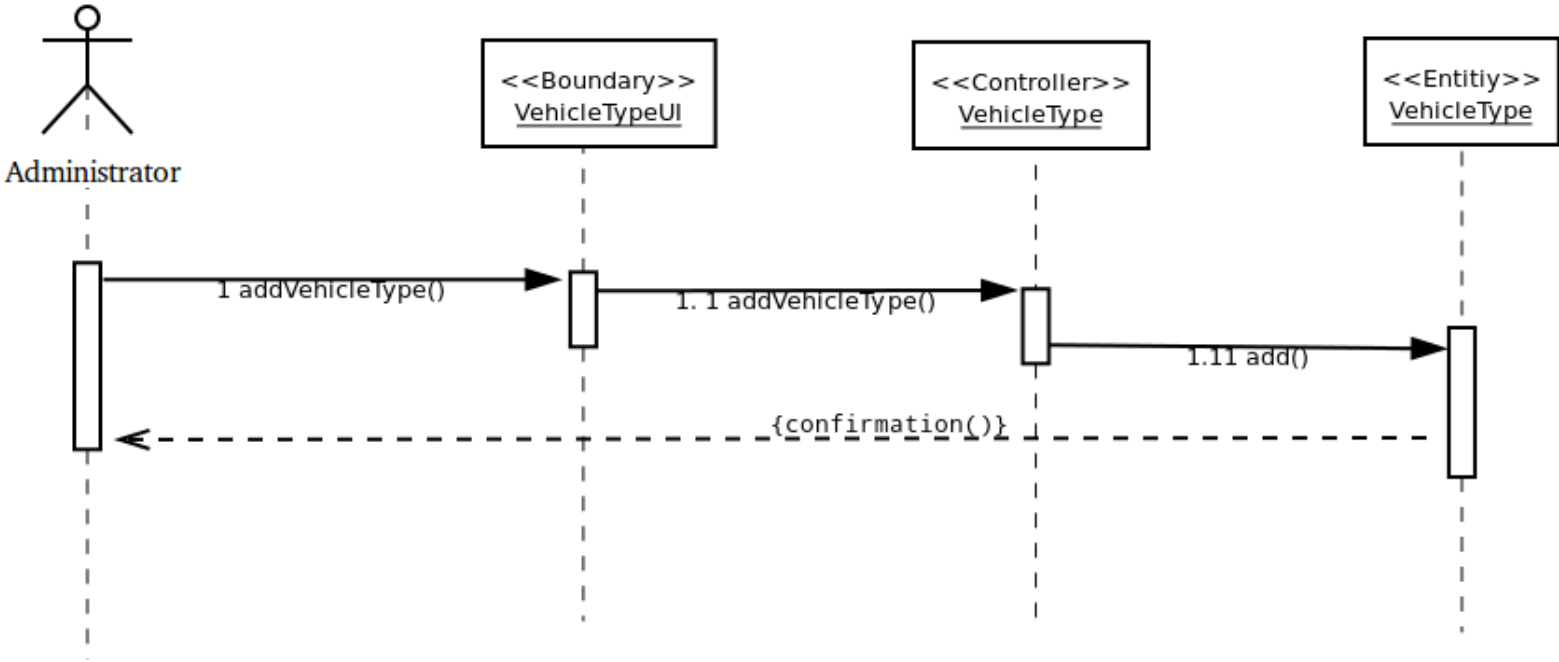
2.1.2.6 DeleteVehicle – Sequence Diagram



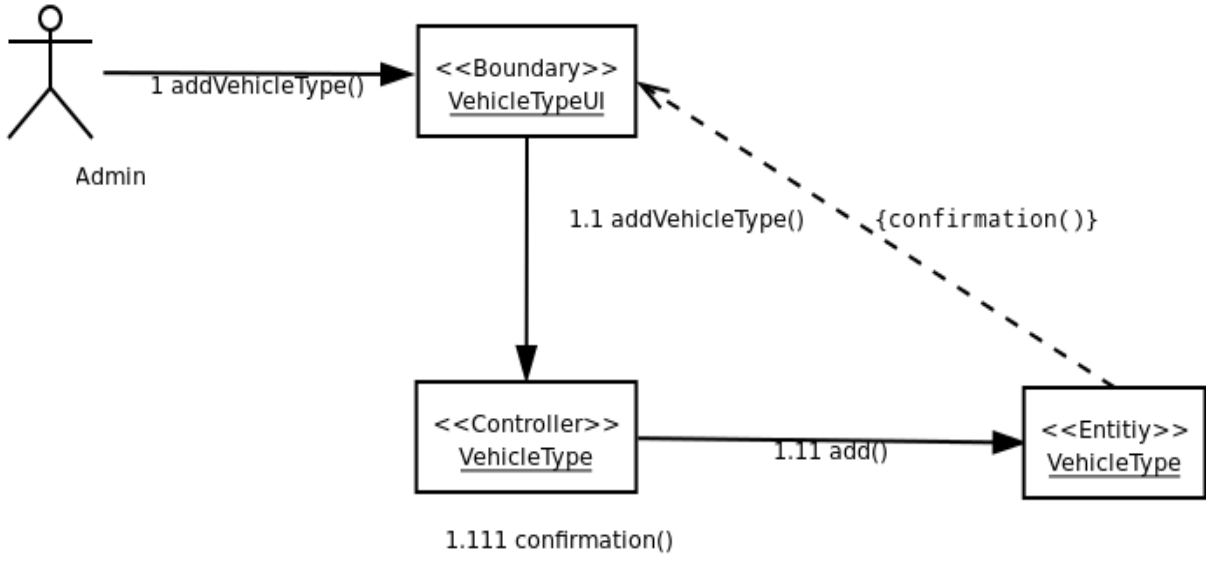
2.1.2.6 DeleteVehicle – Communication Diagra

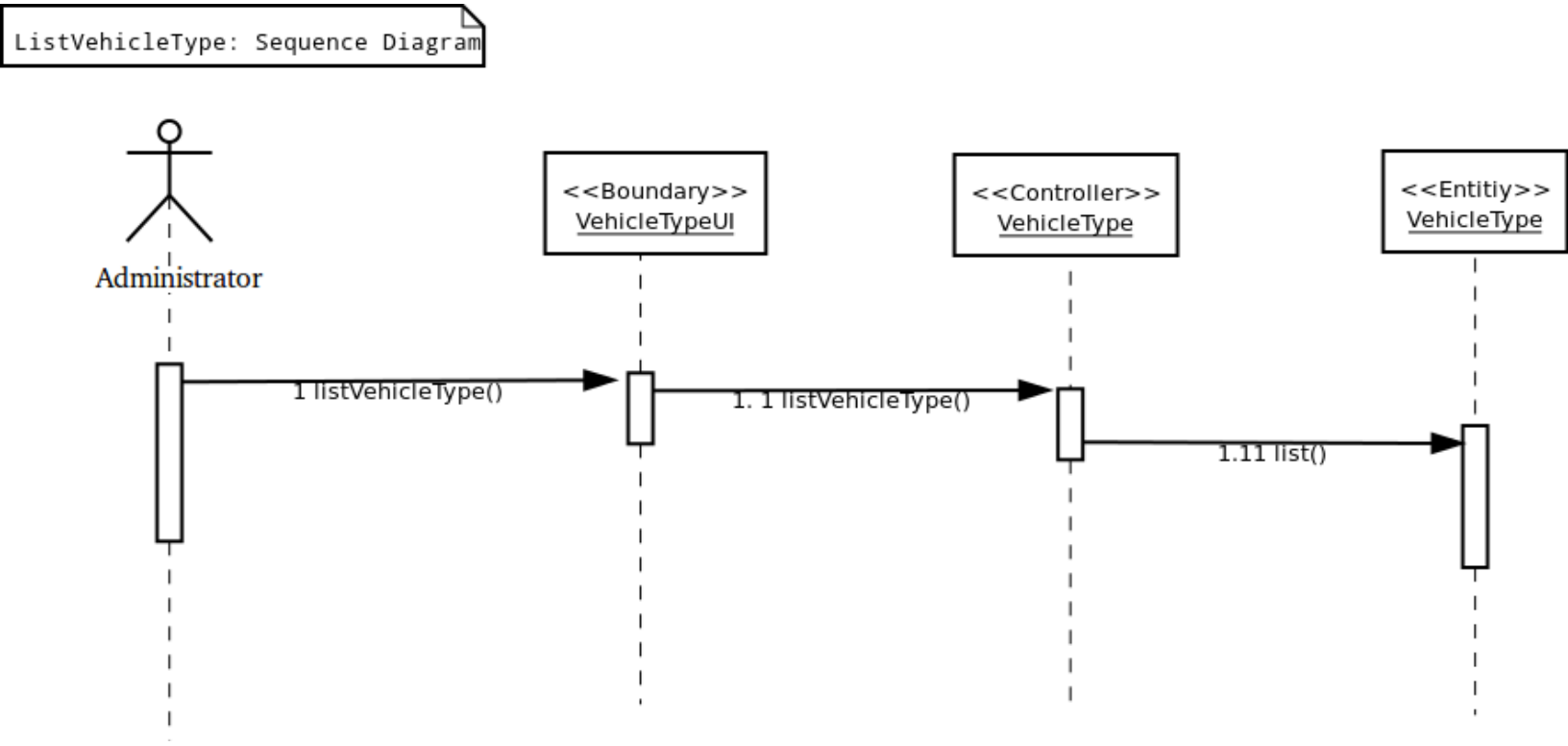


AddVehicleType: Sequence Diagram

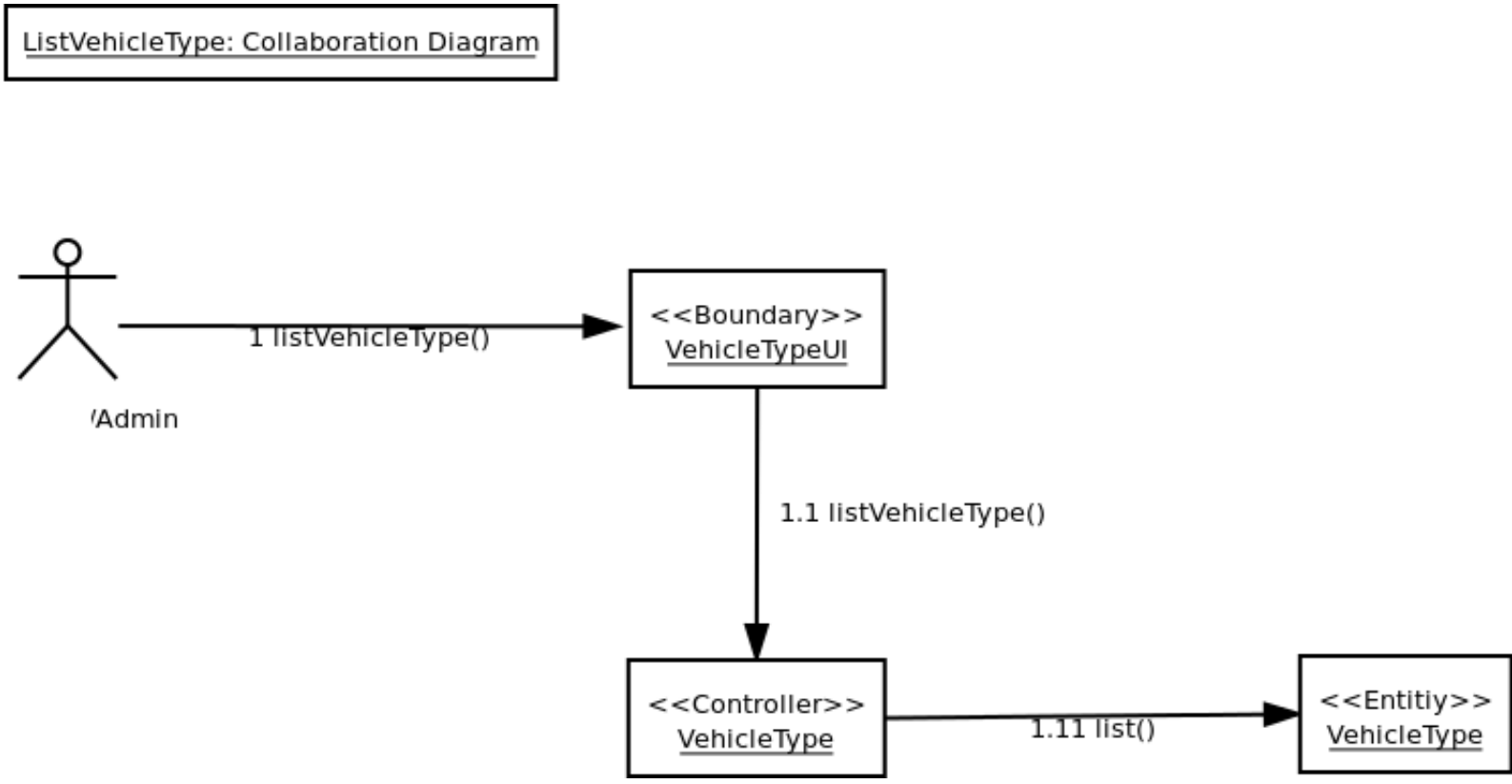


AddVehicleType: Collaboration Diagram

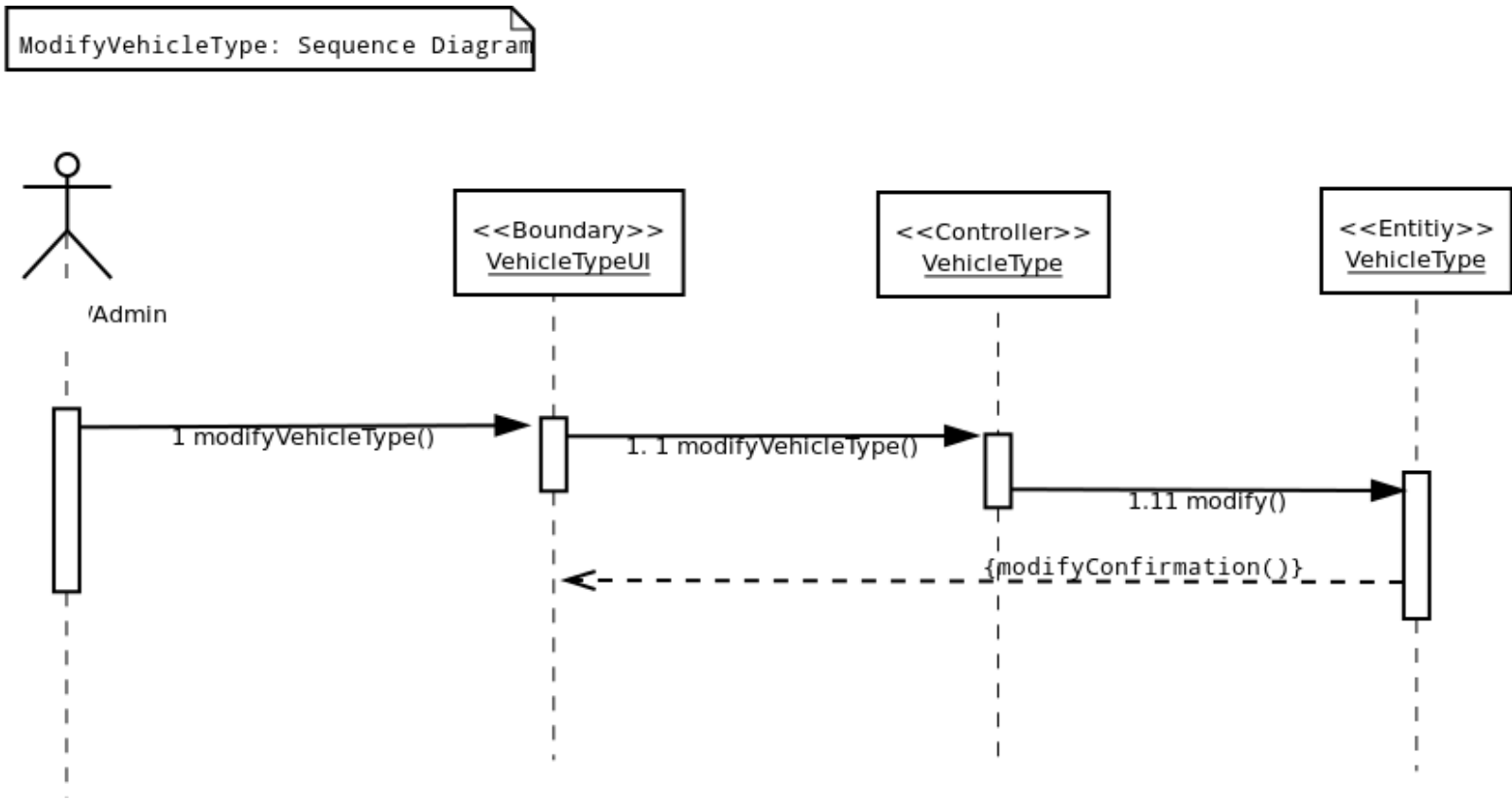




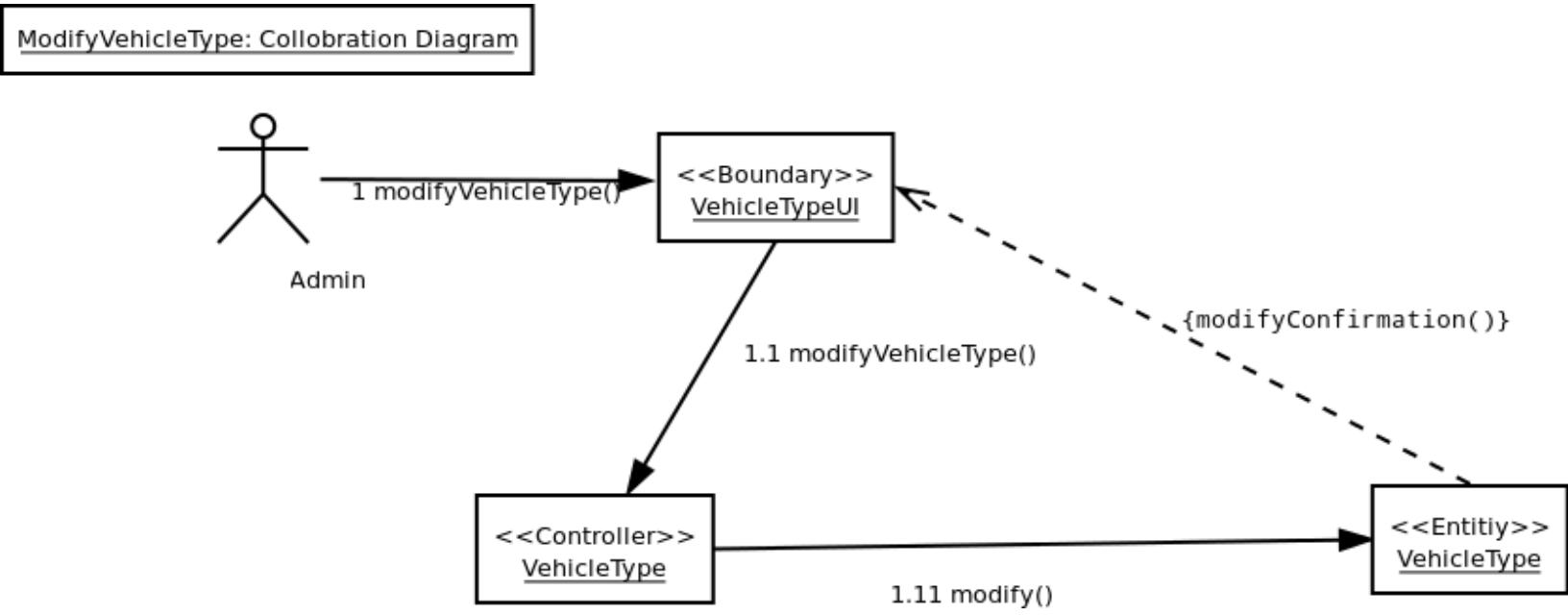
2.1.2.8 ListVehicleType – Communication Diagram



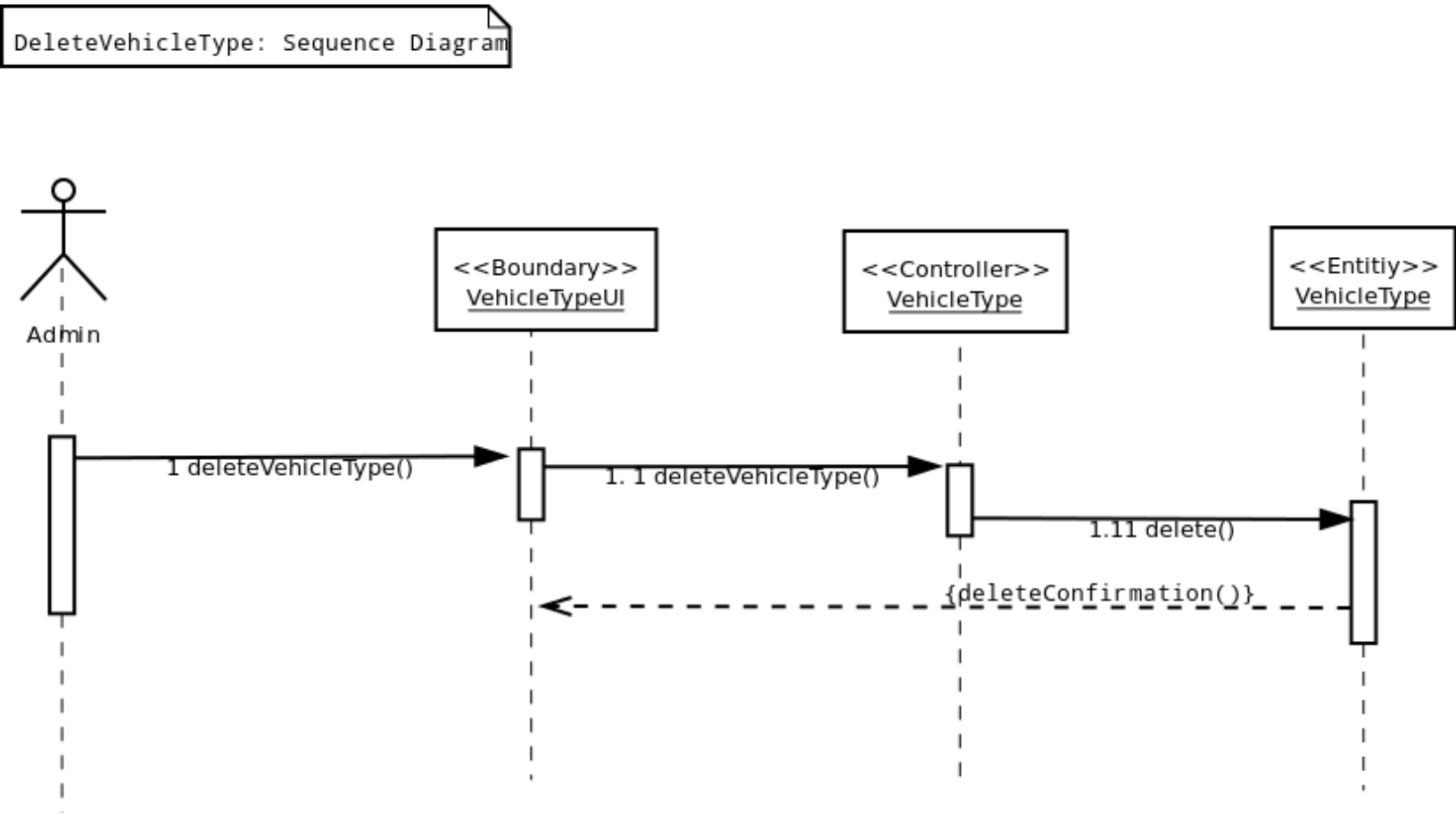
2.1.2.9 ModifyVehicleType – Sequence Diagram



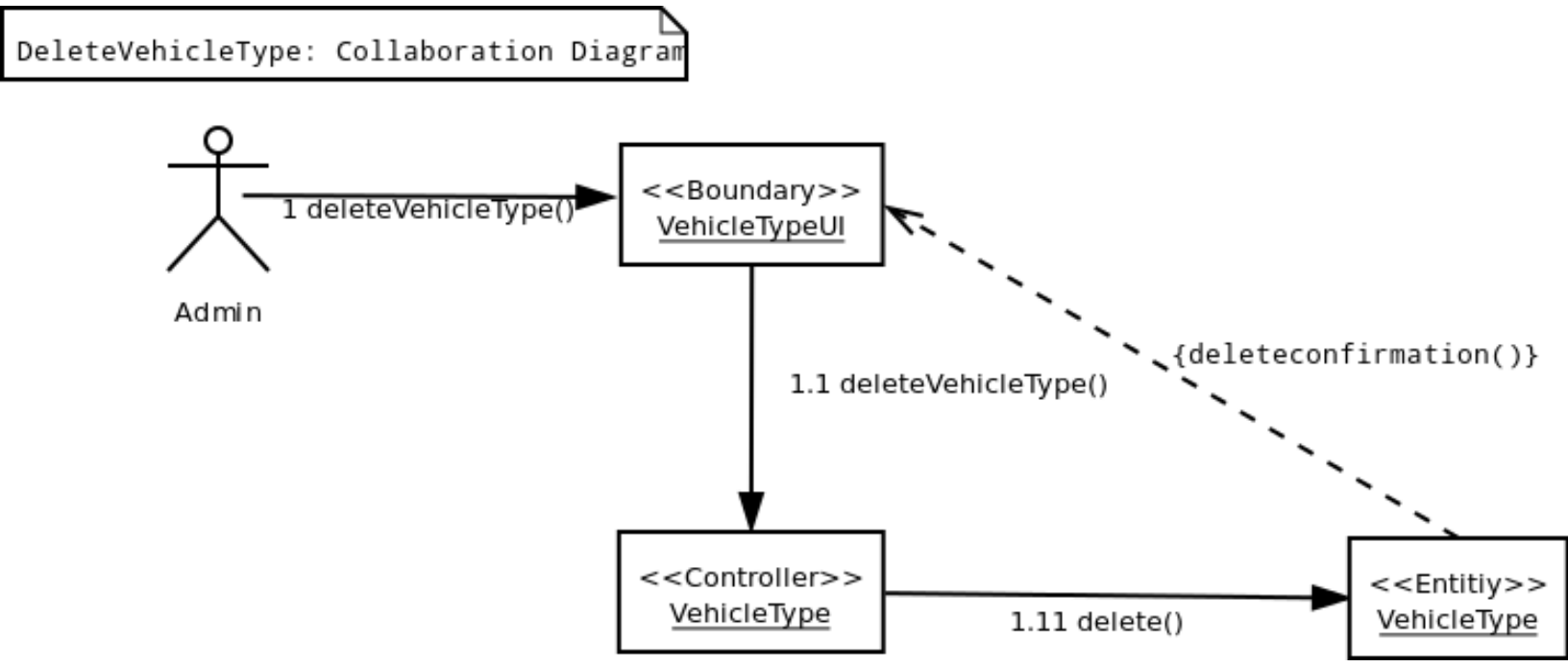
2.1.2.9 ModifyVehicleType – Communication Diagram



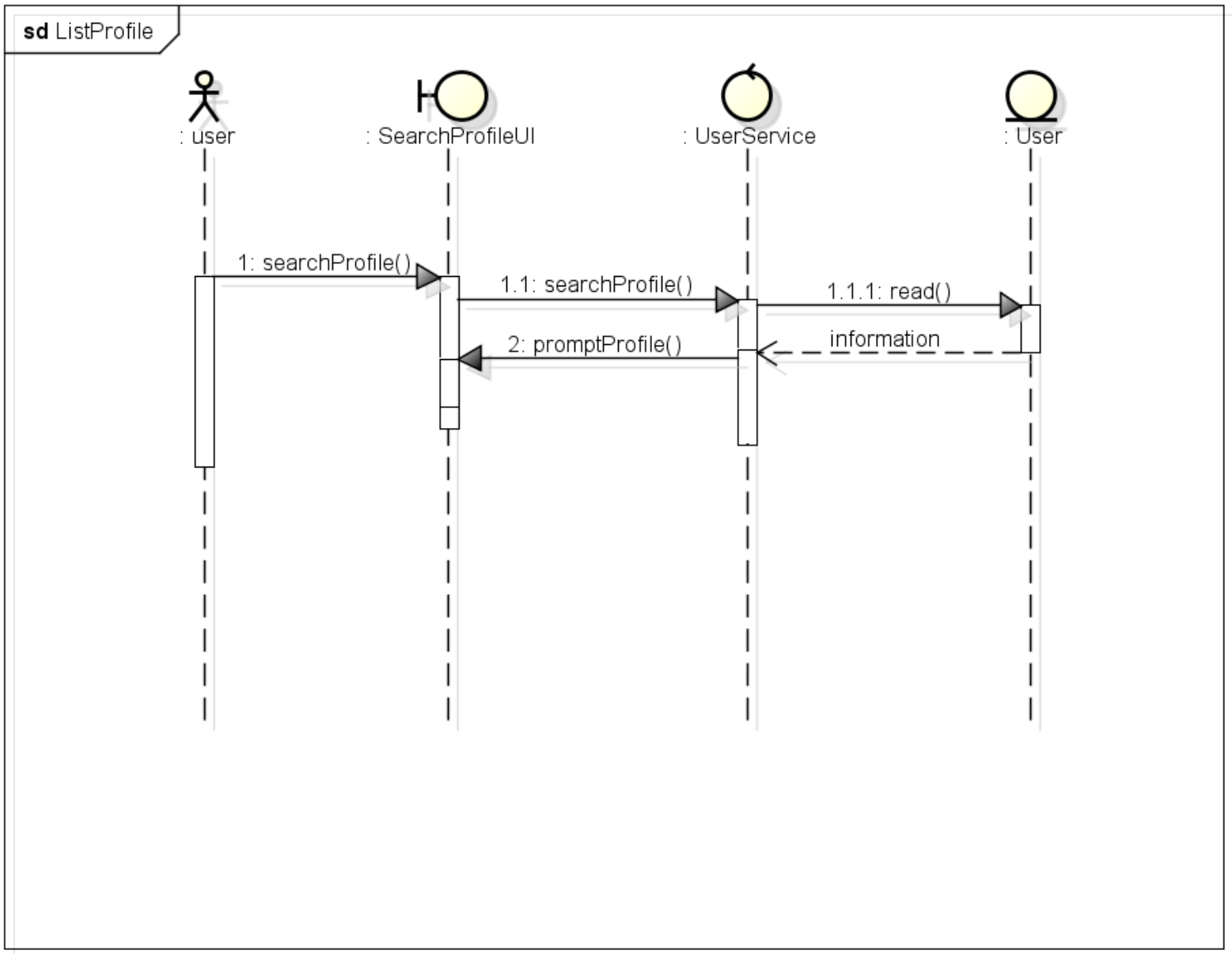
2.1.2.10 DeleteVehicleType – Sequence Diagram



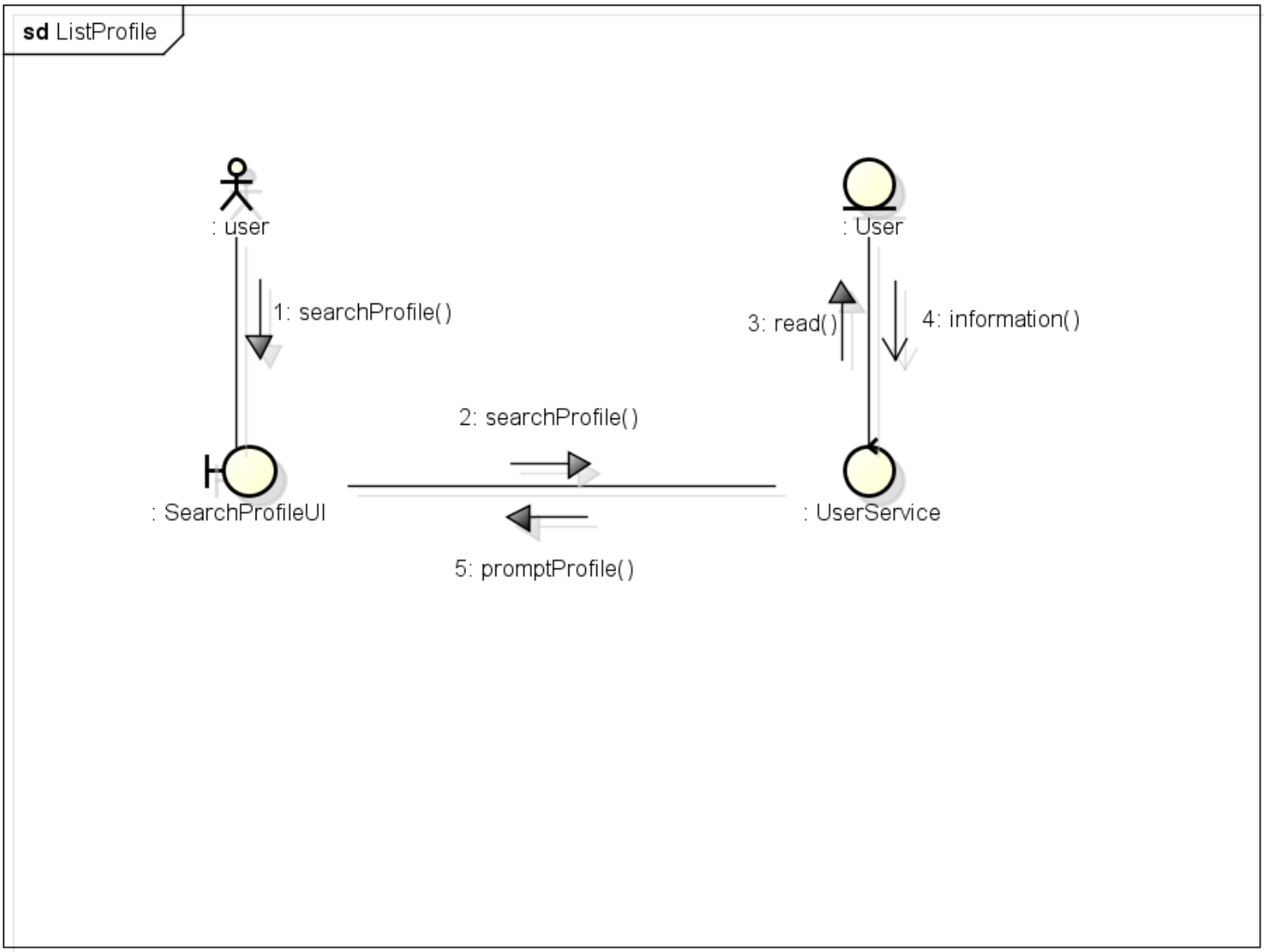
2.1.2.10 DeleteVehicleType – Communication Diagram



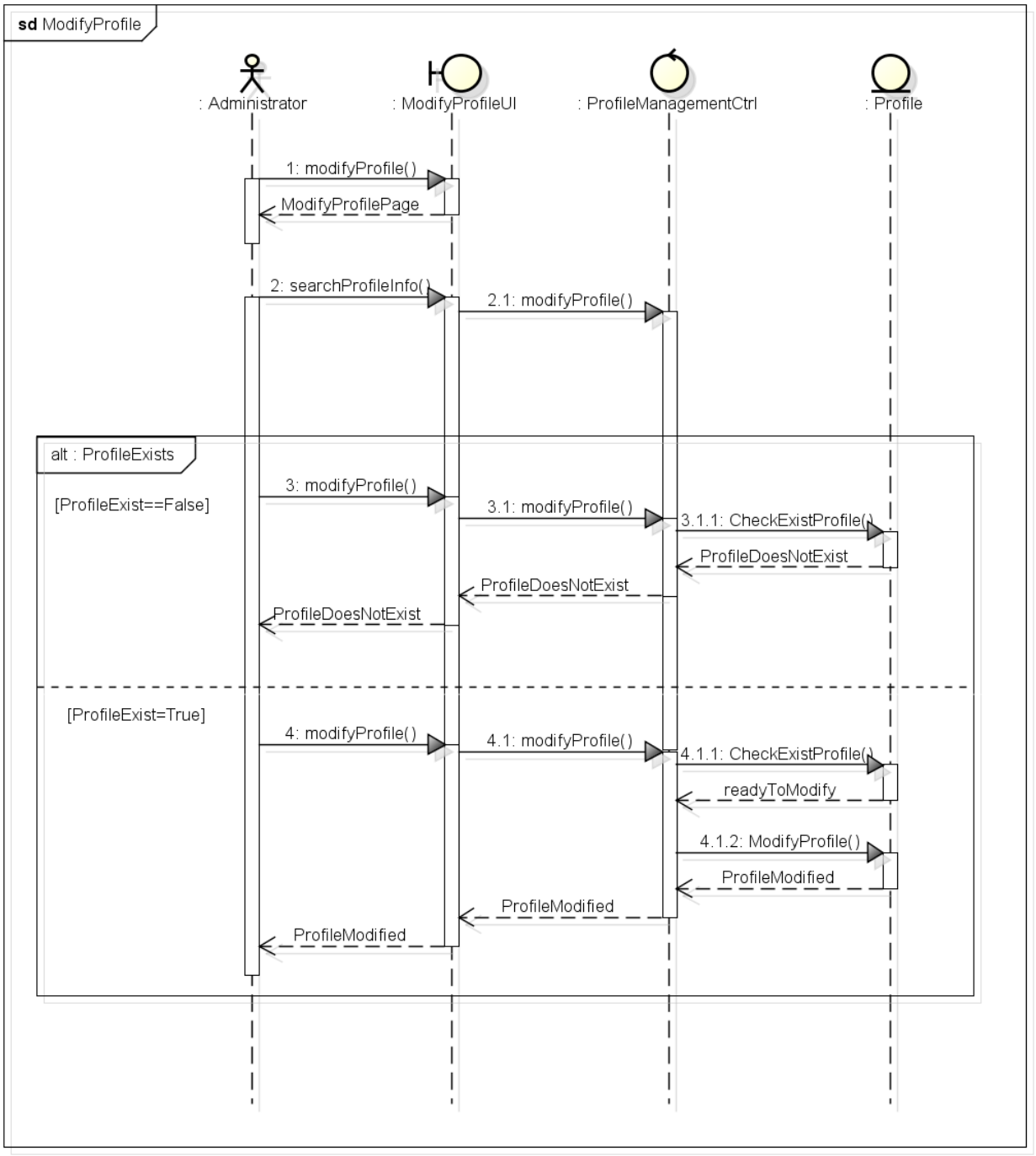
2.1.2.11 ListProfile – Sequence Diagram



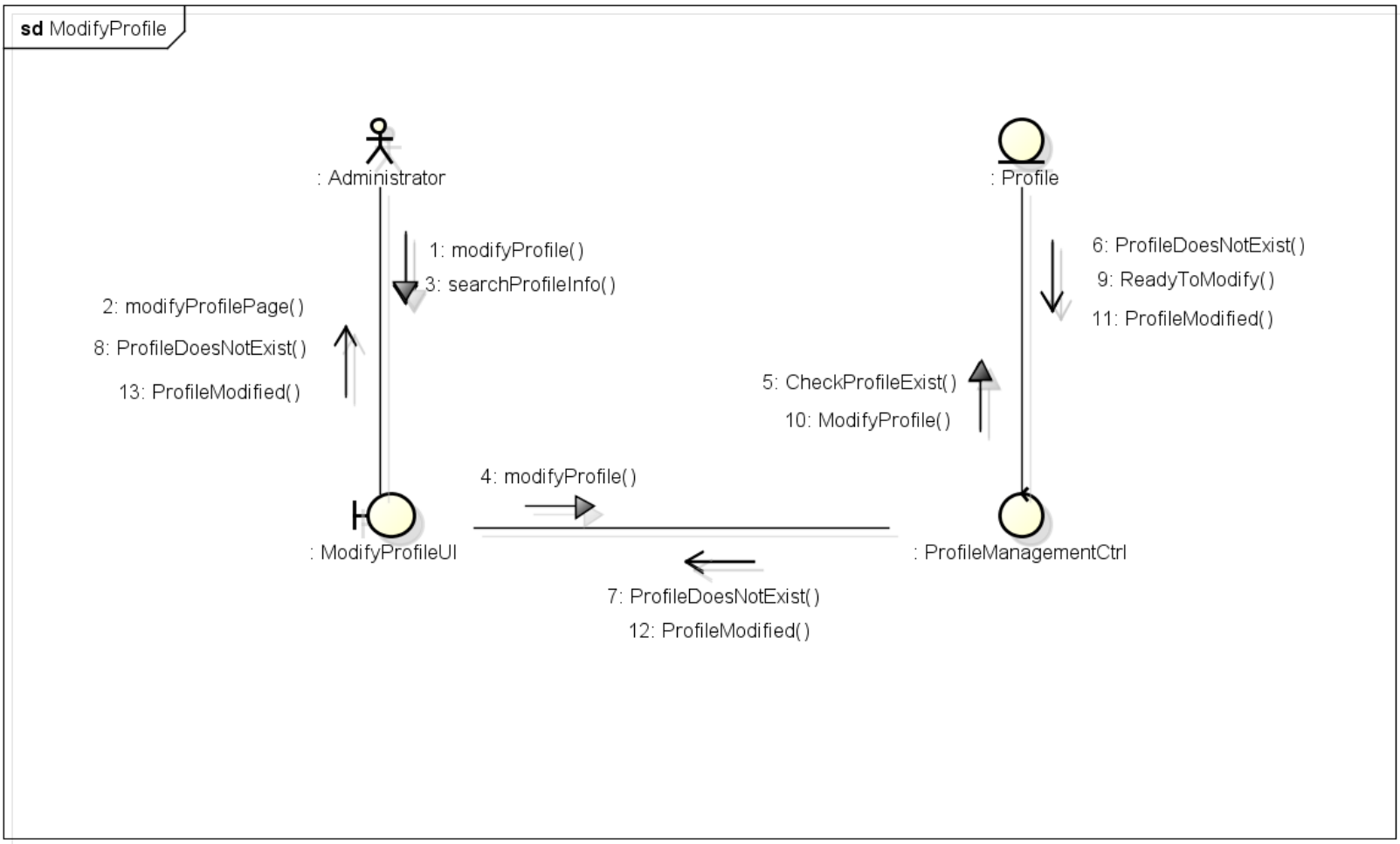
2.1.2.11 ListProfile – Communication Diagram



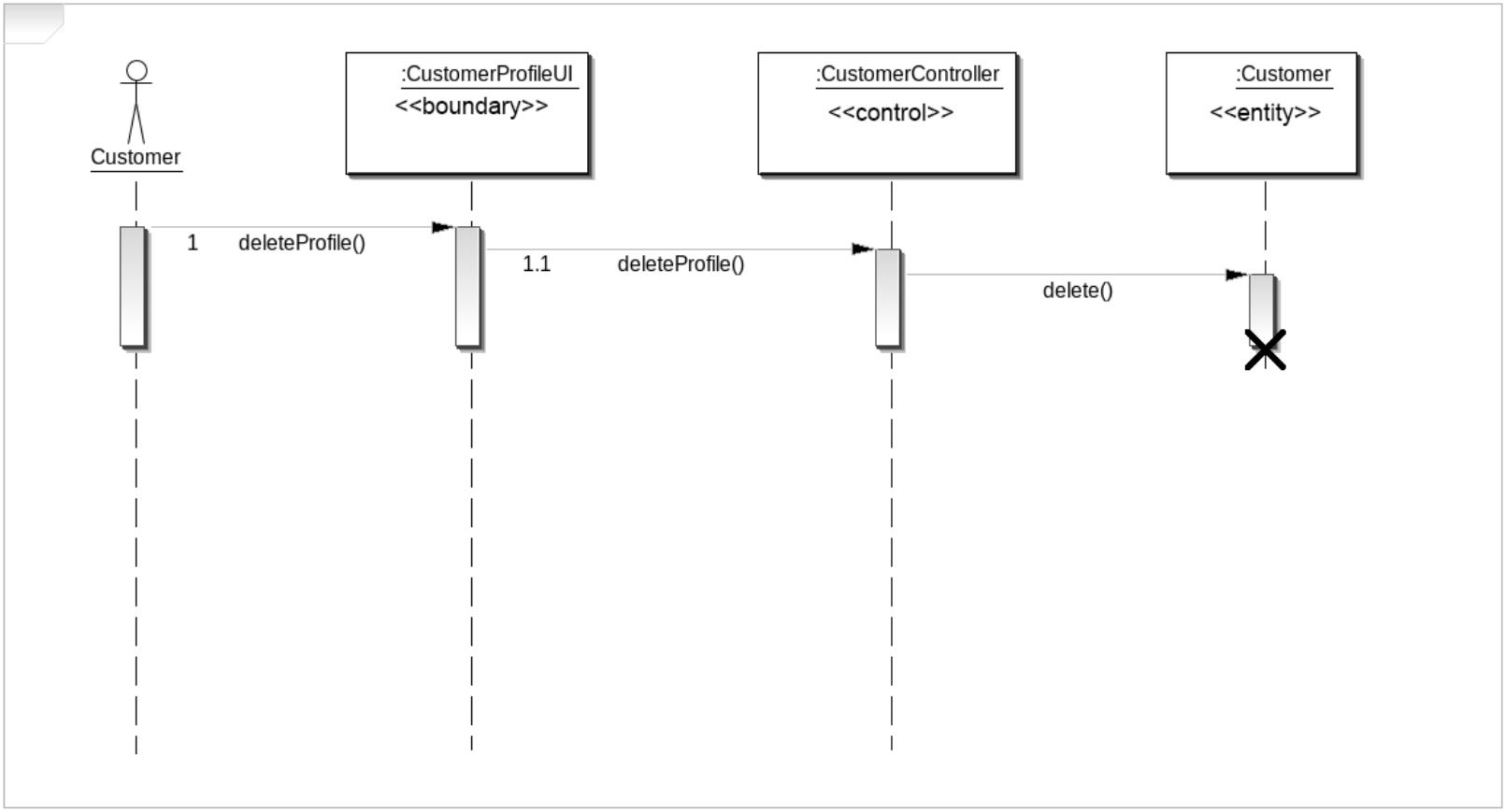
2.1.2.12 ModifyProfile – Sequence Diagram



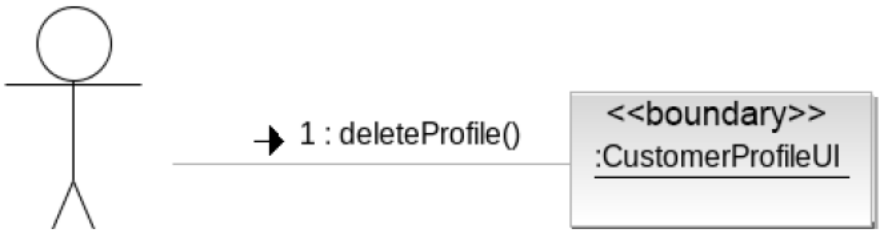
2.1.2.12 ModifyProfile – Communication Diagram

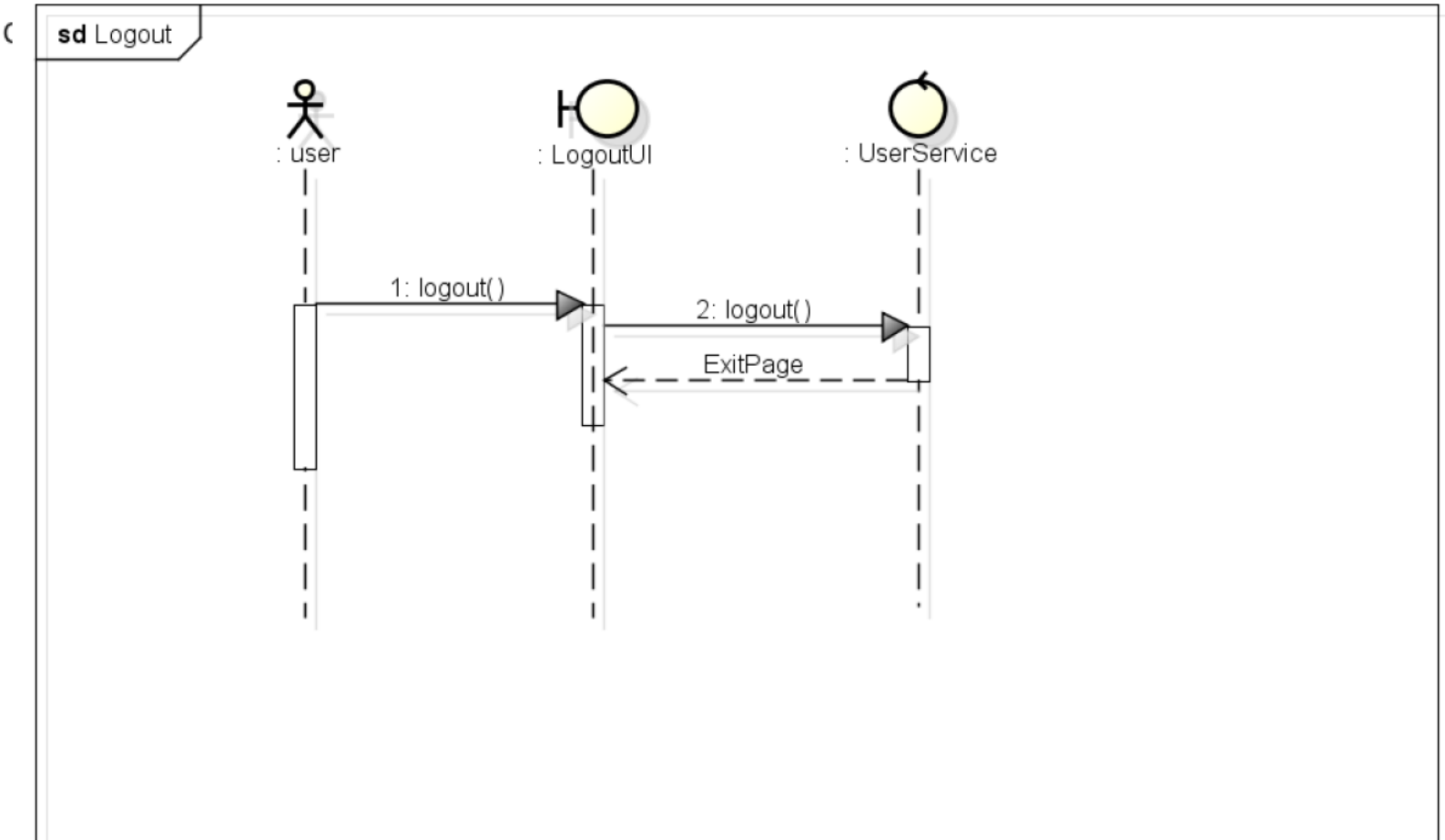


2.1.2.13 DeleteProfile – Sequence Diagram

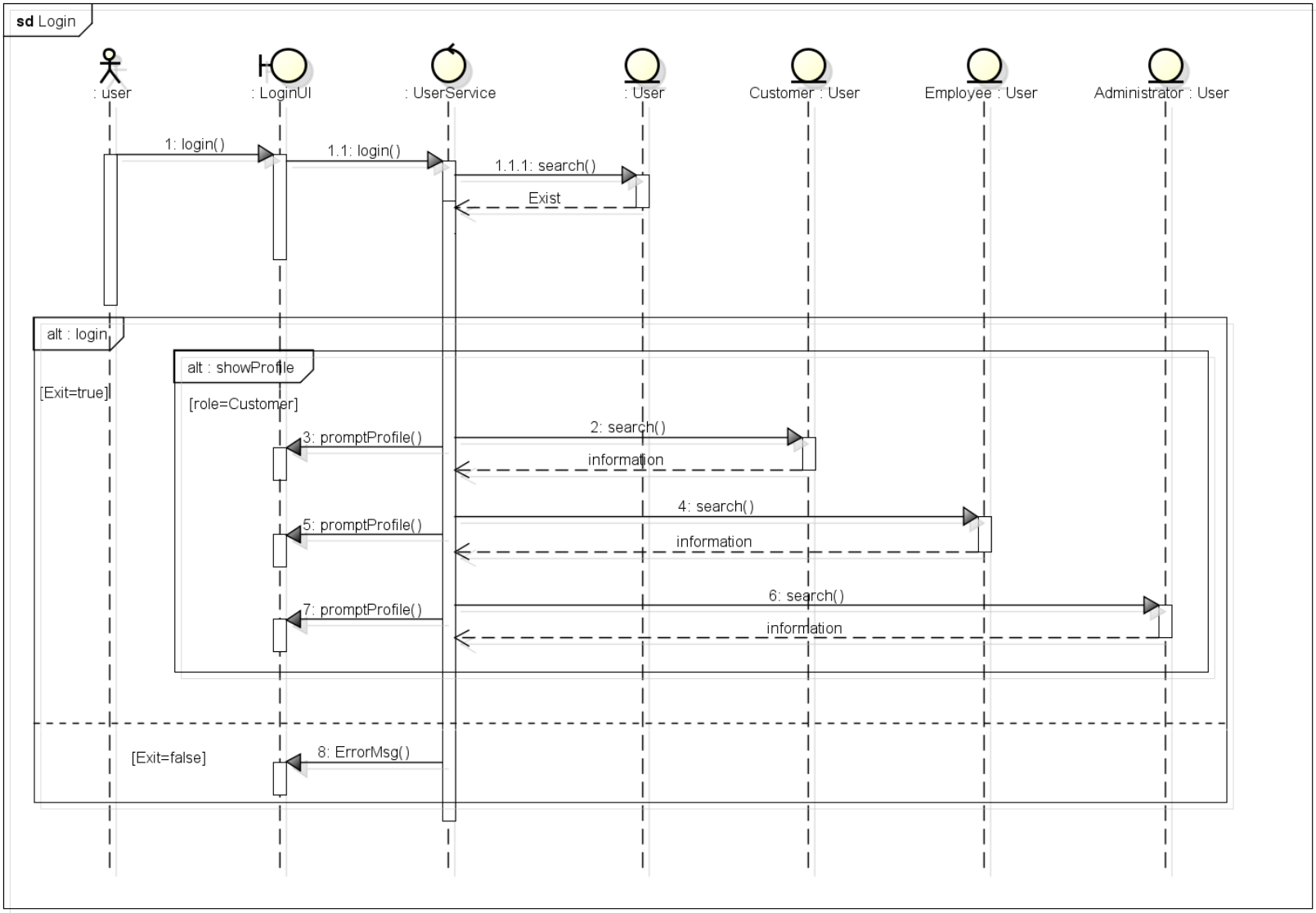


2.1.2.13 DeleteProfile – Communication Diagram

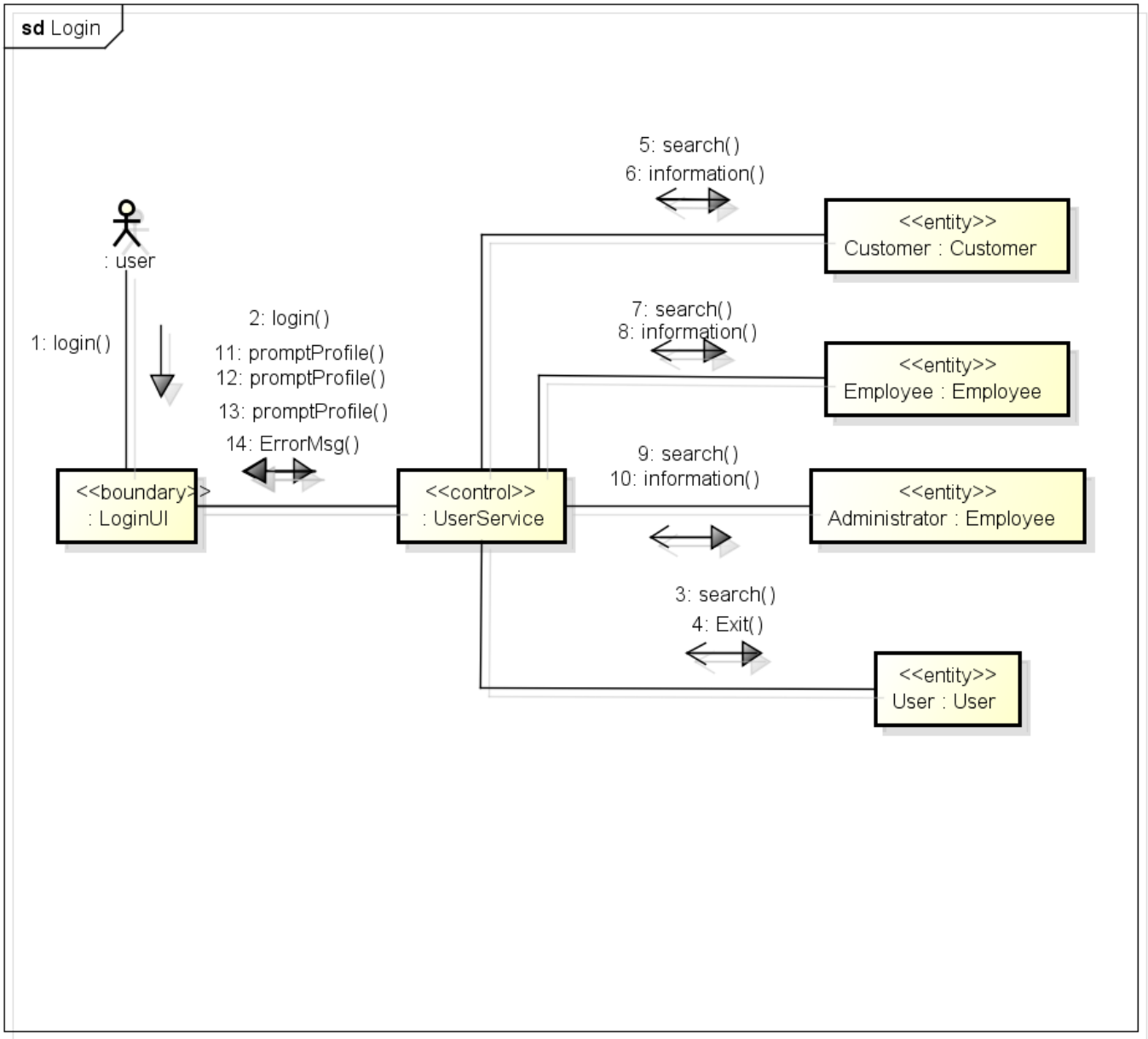




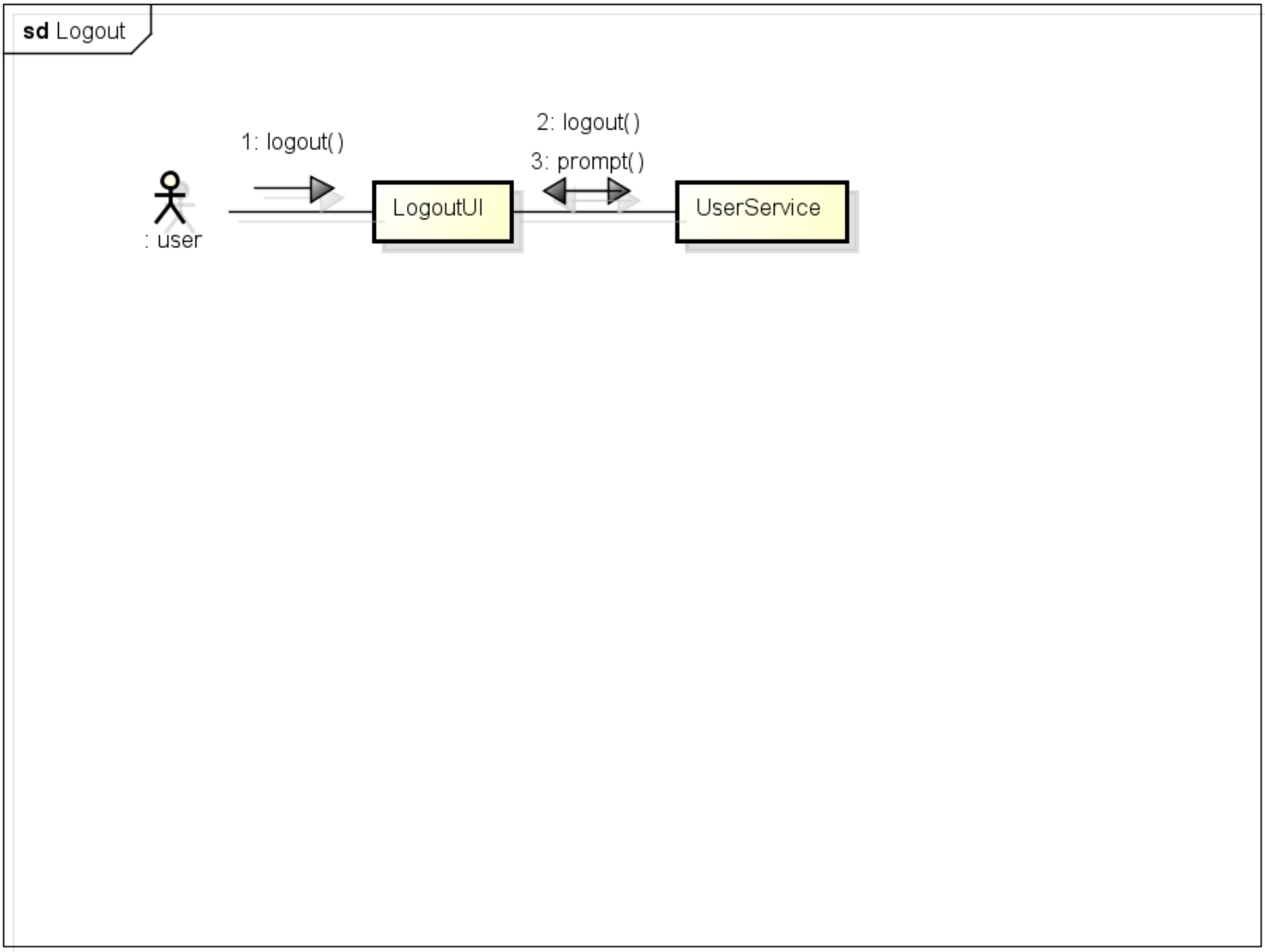
2.1.2.14 Login – Sequence Diagram



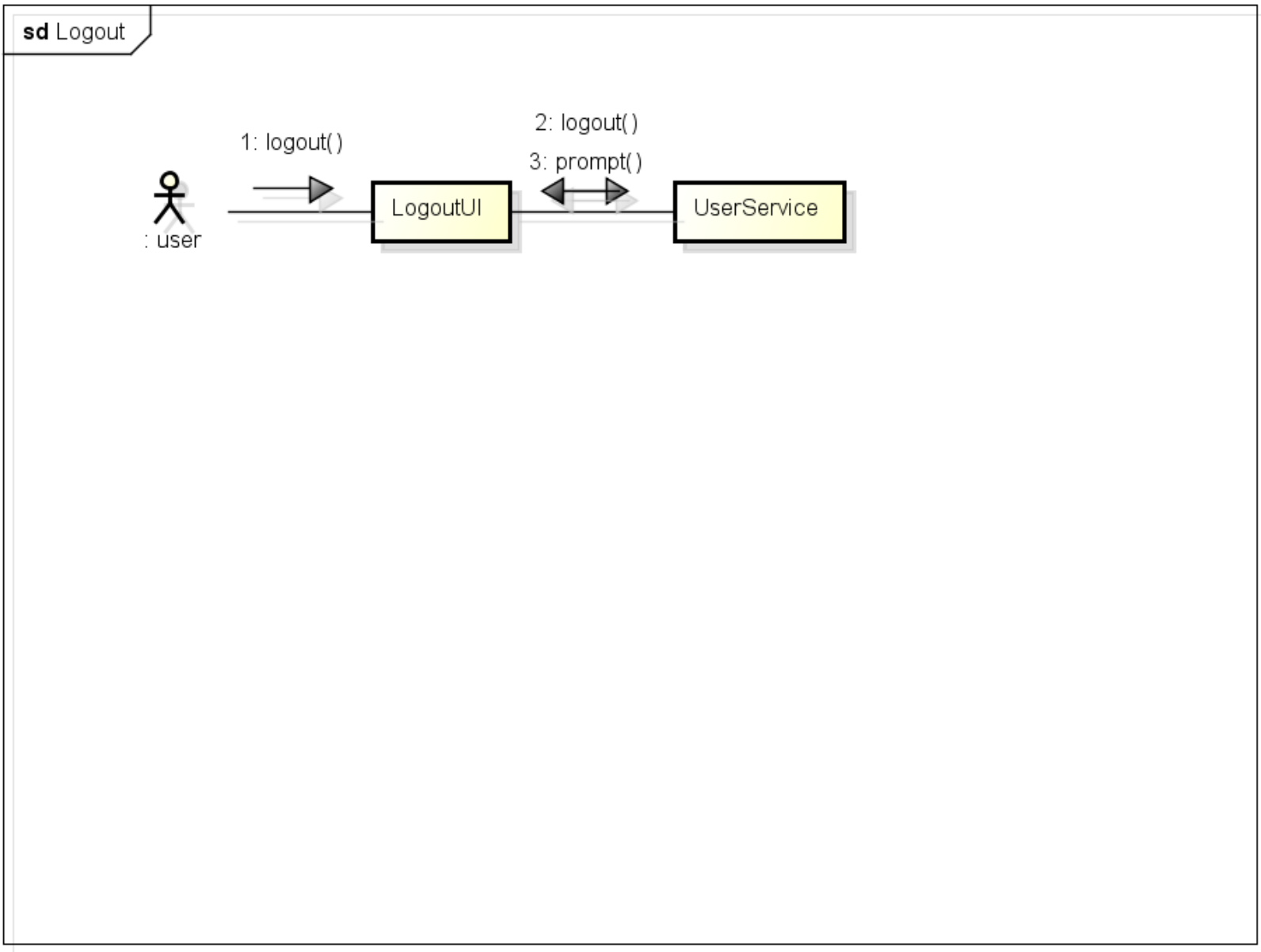
2.1.2.14 Login – Communication Diagram



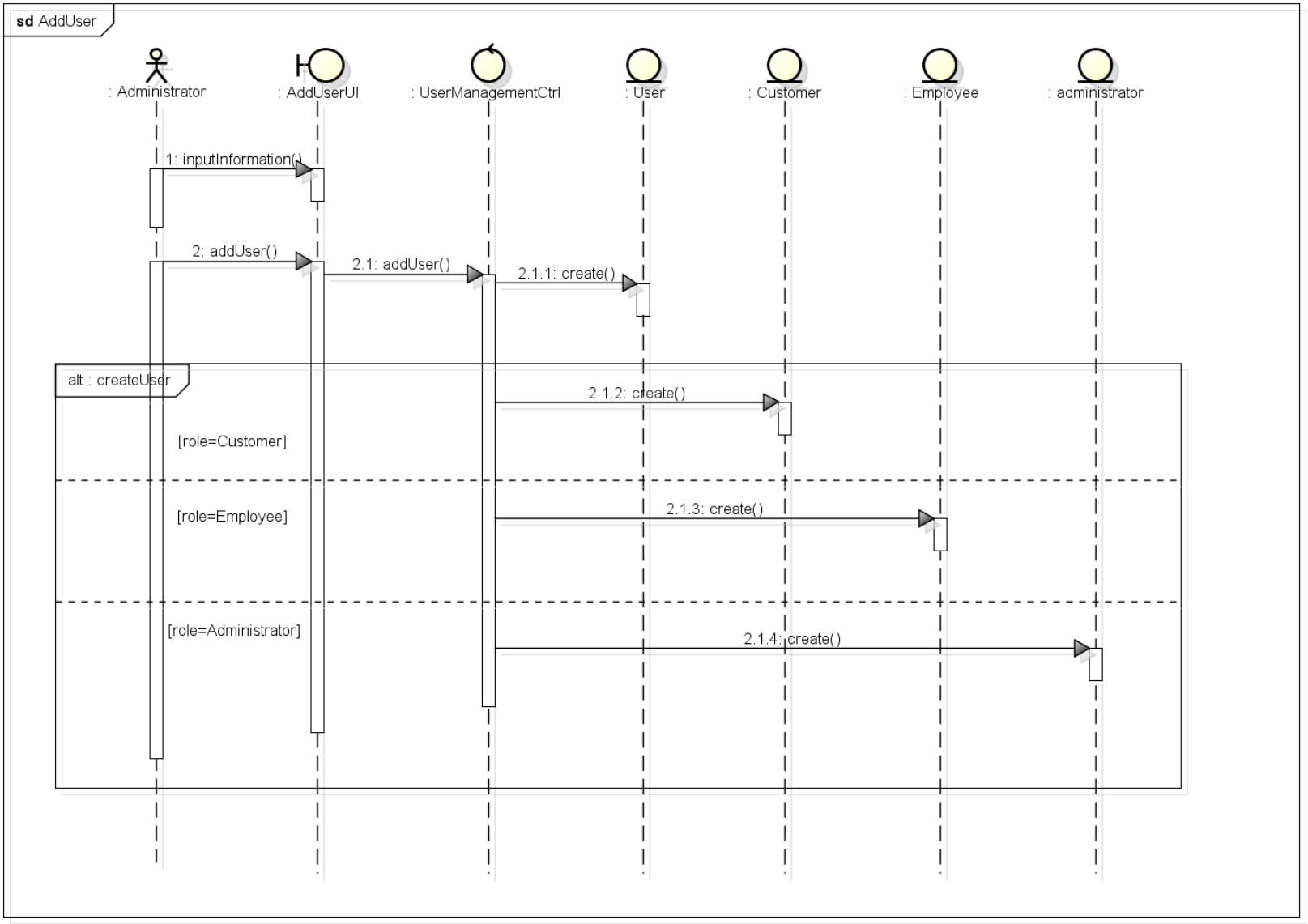
2.1.2.15 Logout – Sequence Diagram



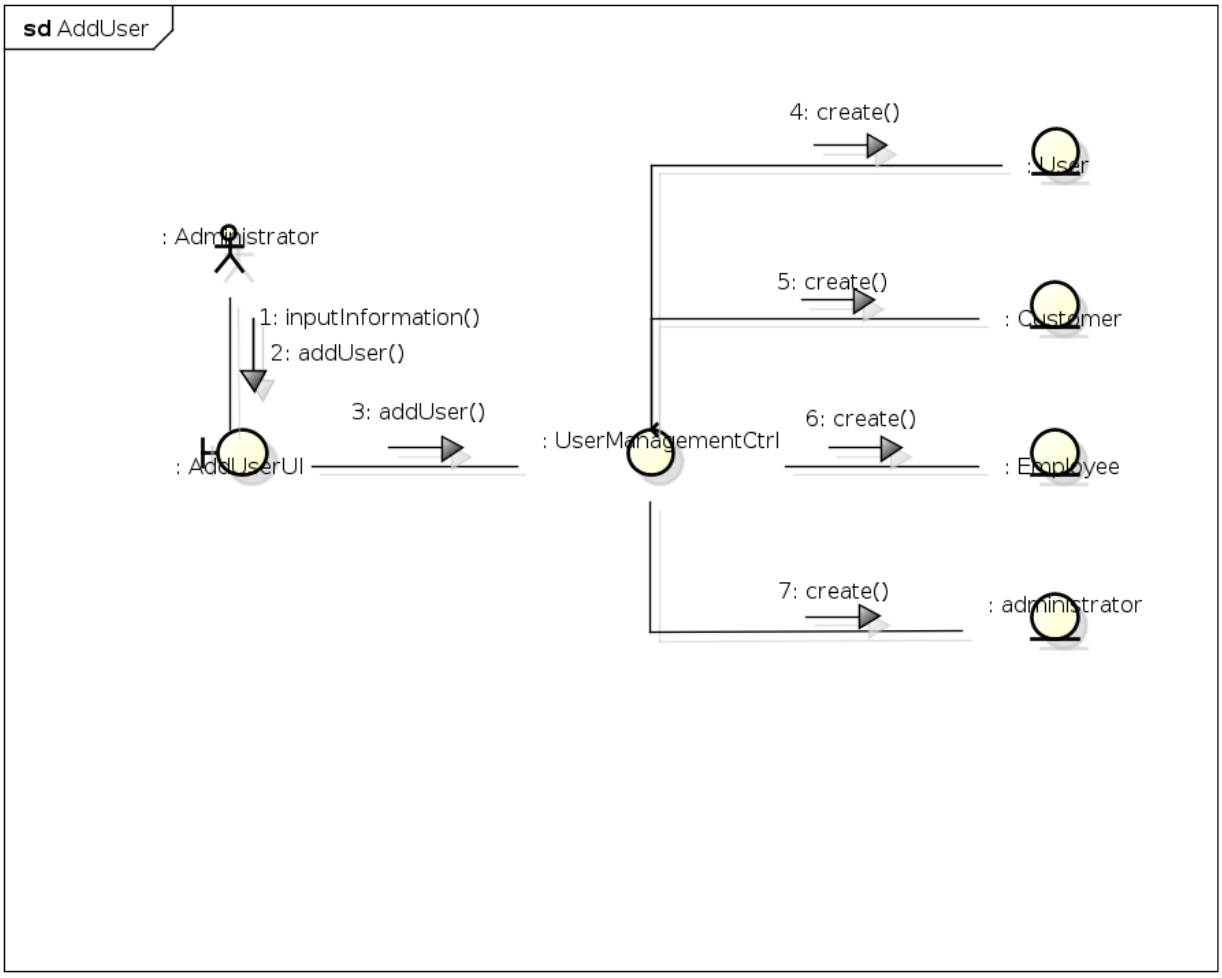
2.1.2.15 Logout – Communication Diagram



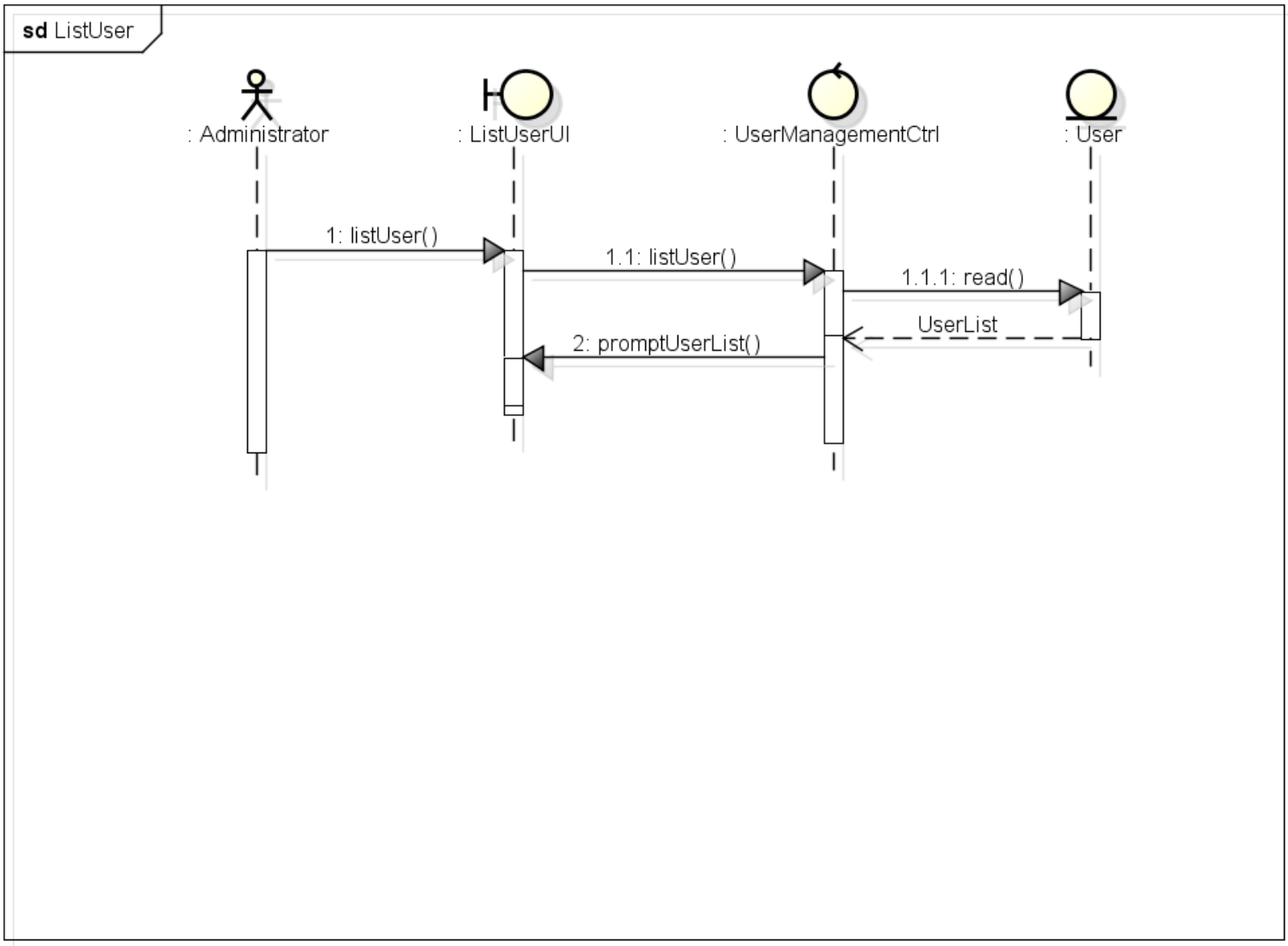
2.1.2.16 AddUser – Sequence Diagram



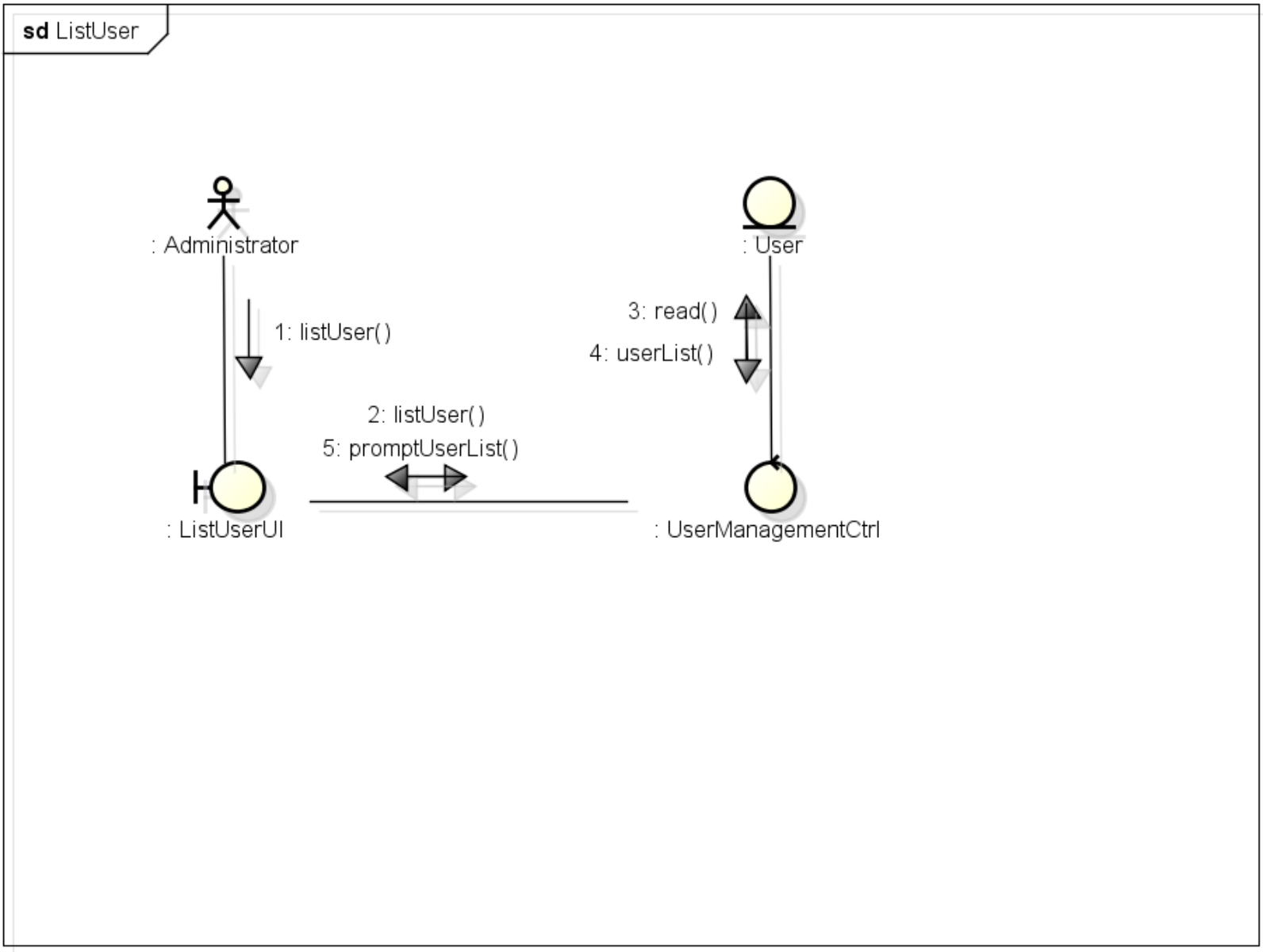
2.1.2.16 AddUser – Communication Diagram



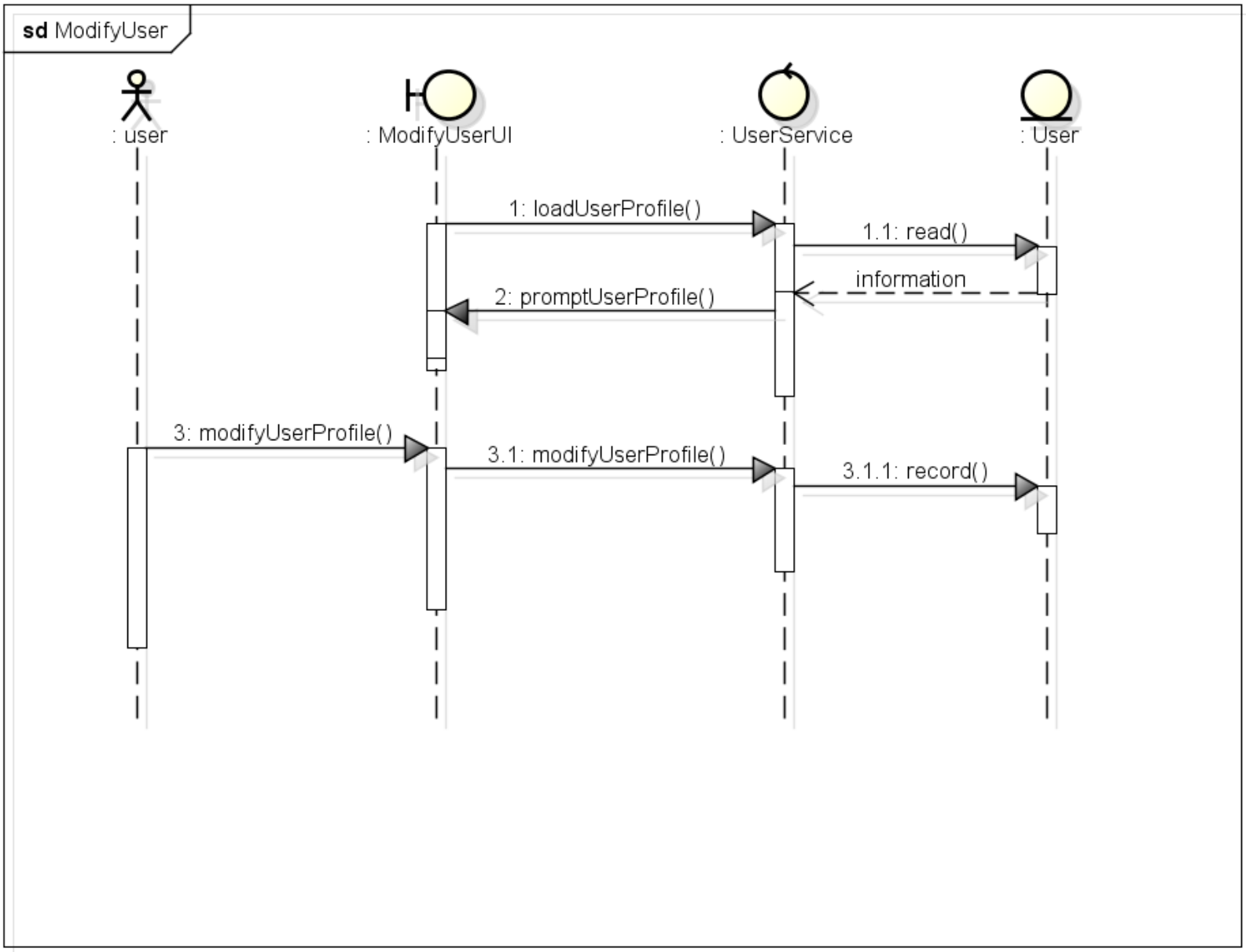
2.1.2.17 ListUser – Sequence Diagram



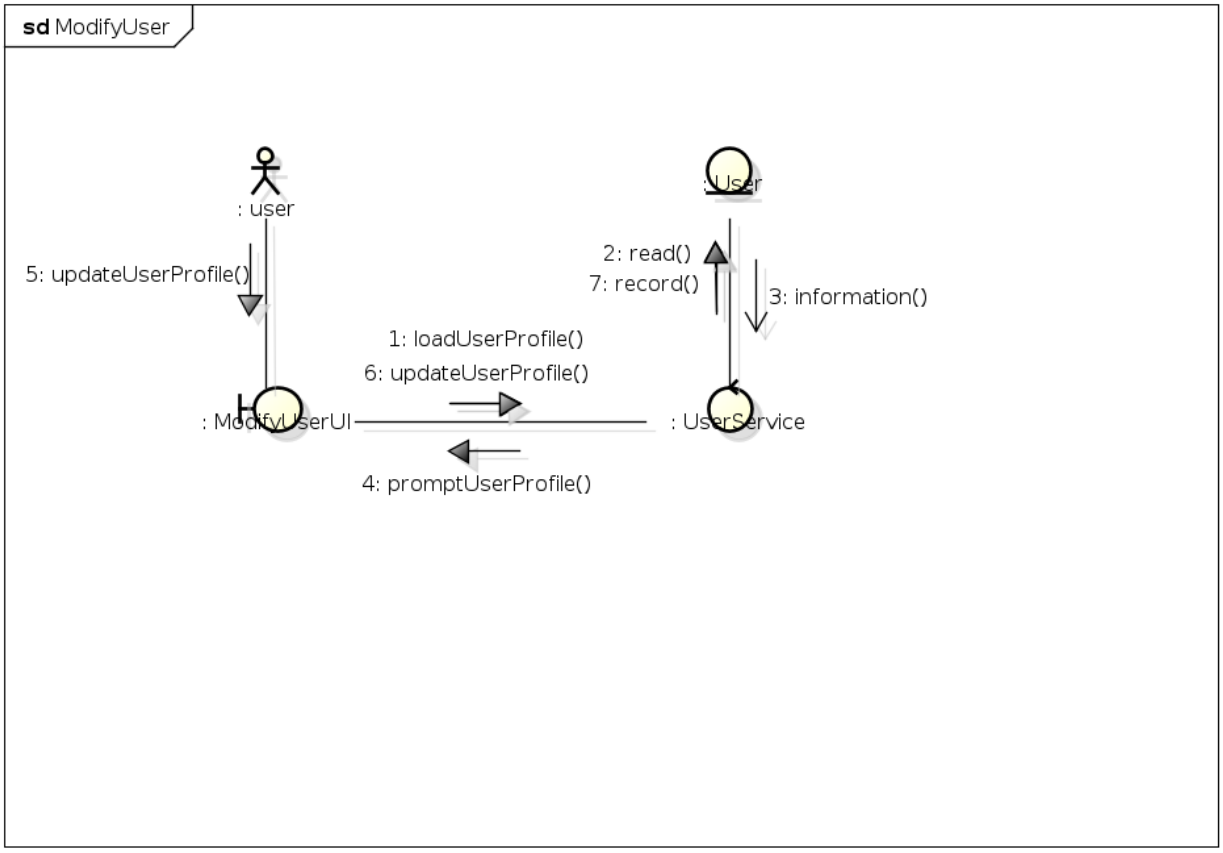
2.1.2.17 ListUser – Communication Diagram



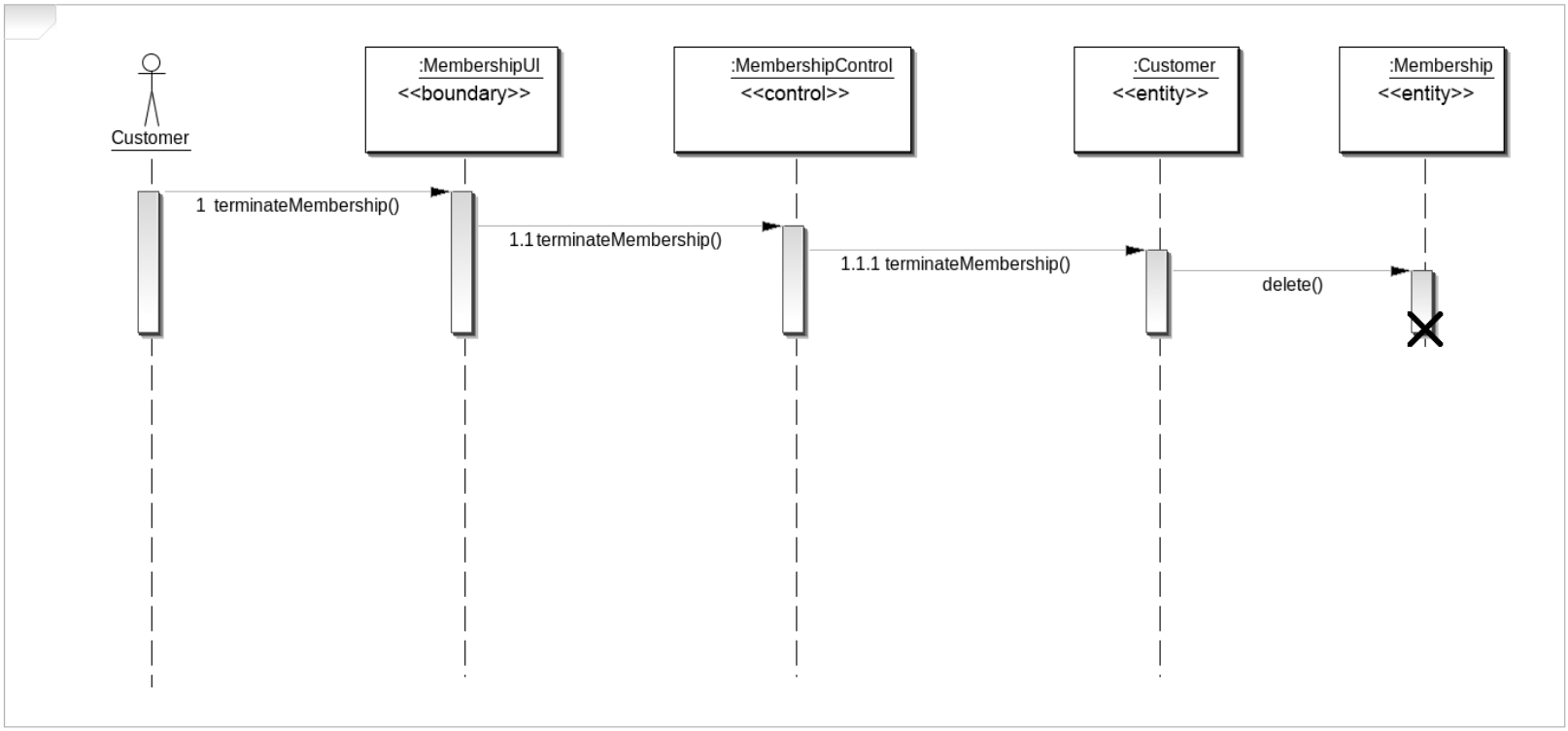
2.1.2.18 ModifyUser – Sequence Diagram



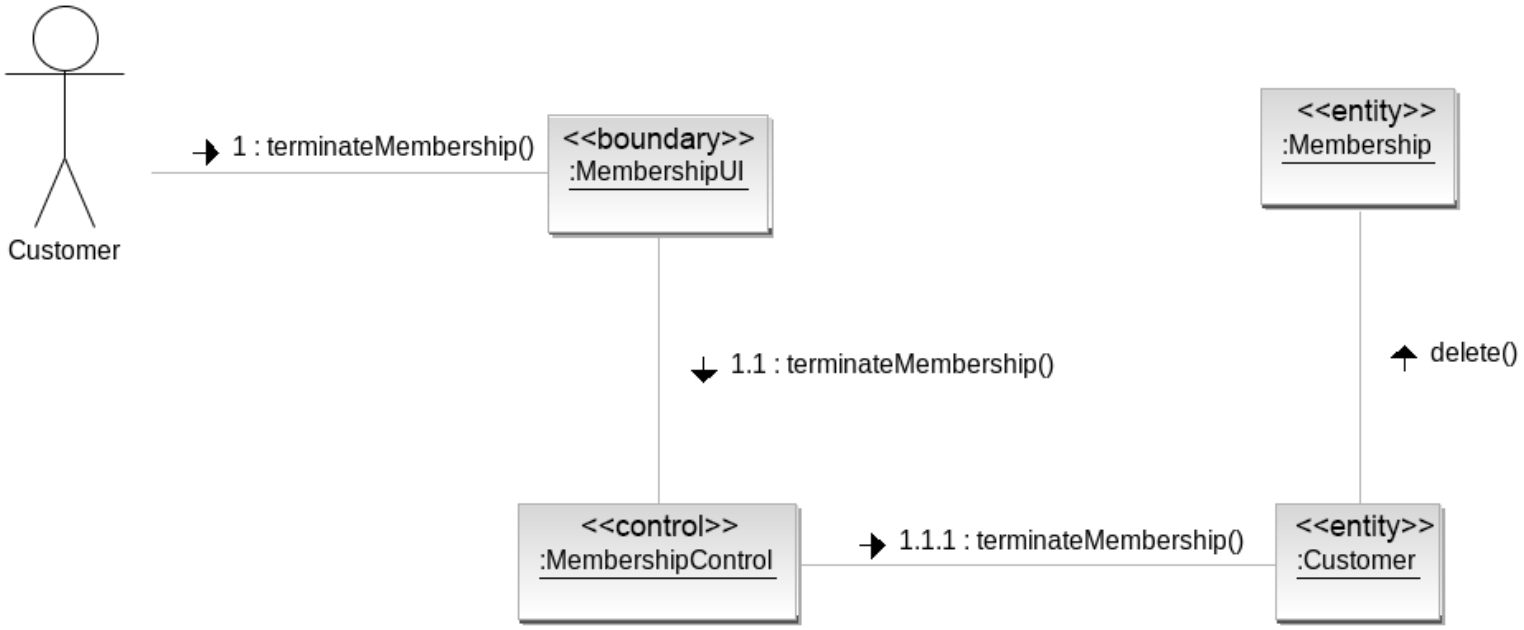
2.1.2.18 ModifyUser – Communication Diagram



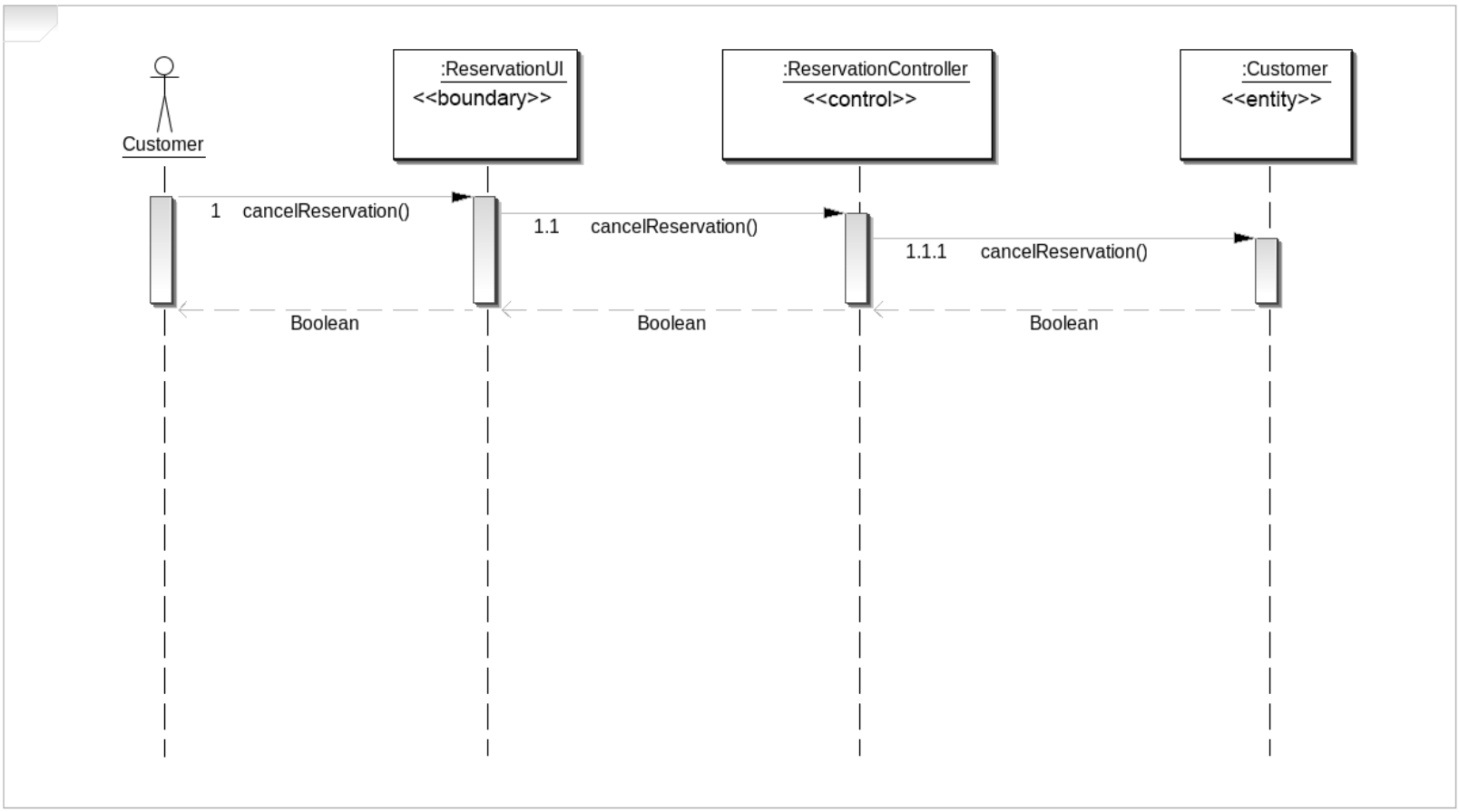
2.1.2.19 TerminateMembership – Sequence Diagram



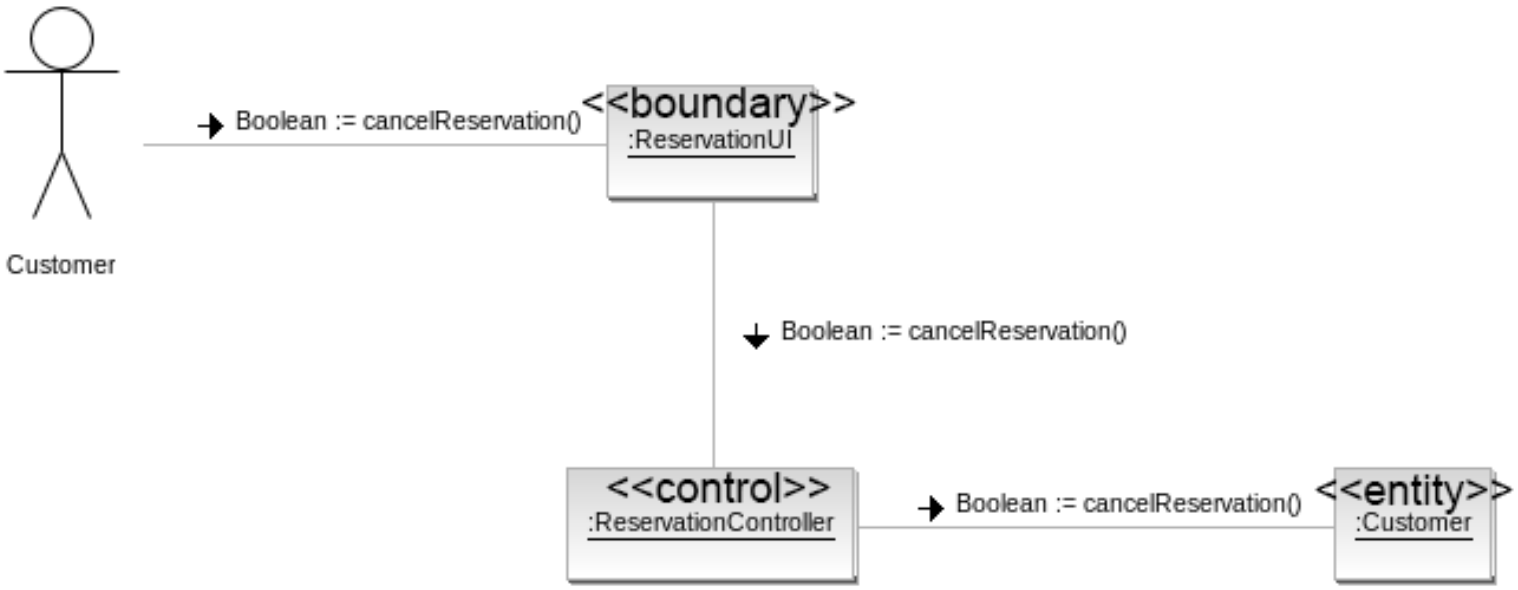
2.1.2.19 TerminateMembership – Communication Diagram



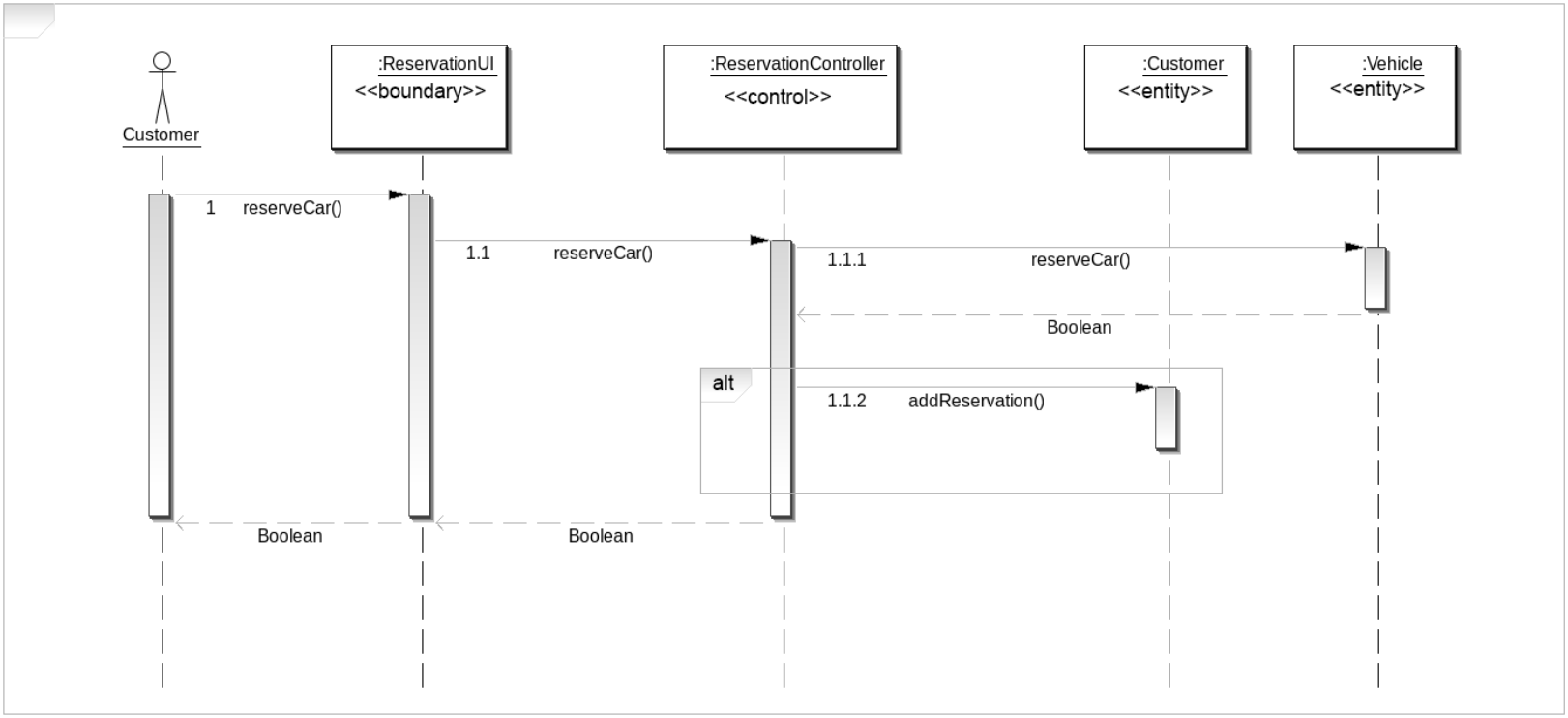
2.1.2.20 CancelReservation – Sequence Diagram



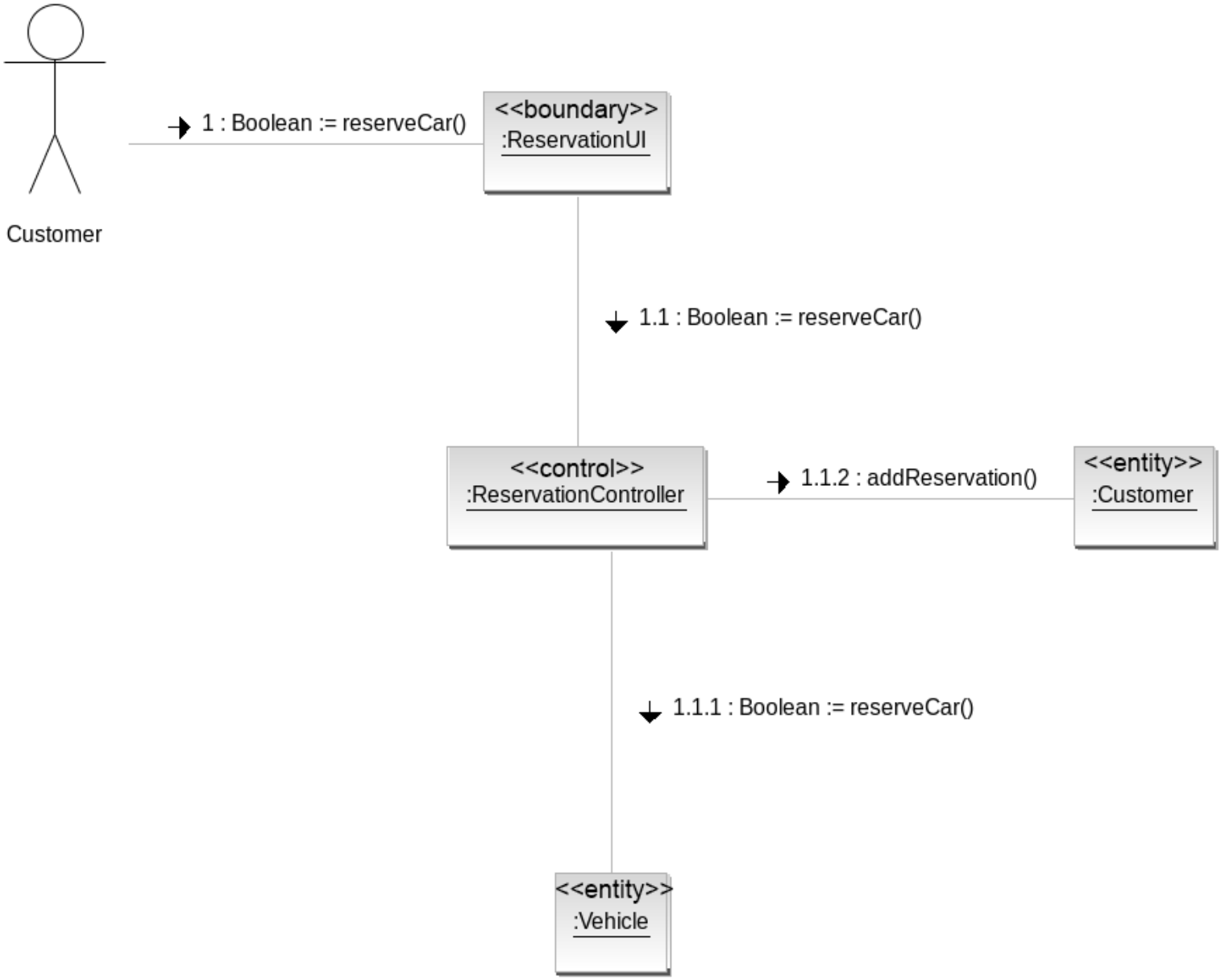
2.1.2.20 CancelReservation – Communication Diagram



2.1.2.21 ReserveCar – Sequence Diagram



2.1.2.21 ReserveCar – Communication Diagram



3. Glossary

Term	Definition
Entity Classes	Model the information handled by the Car rental system. (Ex: Vehicles, Customer Profiles, etc.)
Boundary Classes	Handle communication between actors and system internals. (Ex: Administrator-Database Communication is an example)
Control Classes	Classes that handle the flow of control for our use cases. (Ex: Switching UI display from Search to Search Results)
Control Service Package	A bundle of Control Classes
Boundary Services Package	A bundle of Boundary Classes
Entity Service Package	A bundle of Entity Classes
{User Type}.boundary	Object model we have reference with the following set-up these are a reference to a particular user{administrator or customer} it allows us to specify which set of boundary classes we are addressing. (Ex: administrator .boundary package}
{UserType}.control	Object model we have reference with the following set-up these are a reference to a particular user{administrator or customer} it allows us to specify which set of boundary classes we are addressing. (Ex: administrator.control }
{UserType}.entity	Object model we have reference with the following set-up these are a reference to a particular user{administrator or customer} it allows us to specify which set of boundary classes we are addressing. (Ex: administrator.boundary)
{ActionWord}UI	The Following notation within our notation defines the actual action, and user interface displayed with the associated with our action. This gives our system readability and understanding (Ex: SearchProfileUI, VehicleTypeUI, etc).

Object Design Document:
Car Rental Service

Group 4
CSCI 4050: Software Engineering
Instructor: Krys Kochut
Osama Mansour, Stephen Patton, Vincent Lee, and Minh Pham

Table of Contents

1. Introduction

The online car rental service will be implemented through various subsystem layers that interact with each other. In this object design document, we specify a complete blueprint of the system we will be implementing, finalizing intricate details of classes, methods, and subsystems.

1.1 Object Design Trade-Offs

1.1.1 Development Cost vs. Functionality

The system is designed to include a large amount of functionality. Vehicles can be checked in an out online, and system administrators can add, delete, and modify all vehicles, vehicle types, and rental car store locations. Our vehicle search can be done based on many different parameters including location, vehicle type, and price. We feel that these features, although expensive to implement, are necessary to provide adequate functionality to the customer and administrators.

1.2 Interface Document Guidelines

1.2.1 Naming Conventions

Our implementation will follow standard Java naming conventions:

packages

lowercase

files

same name as the public class identifier

classes

capitalize the first letter and the first letter of any internal words

constants

all uppercase with underscores separating words

methods

first letter lowercase, remaining words begin with a capitol

comments

classes will begin with “/*.....*/” with a short description of the functionality of the class and instructions for usage. The following tags may be included for additional documentation:

1. @author author-name
2. @version version number of class
3. @see string
4. @see URL
5. @see classname #methodname

Within methods, // may also be used to give additional comments when needed.

methods

Javadoc conventions will be followed when outlining the purpose, usage, return values, parameters, exception, etc. of a method. This includes the following tags:

1. @param paramName description
2. @return description of return value
3. @exception exceptionName description
4. @see string
5. @see URL
6. @see classname#methodname

1.3 Definitions

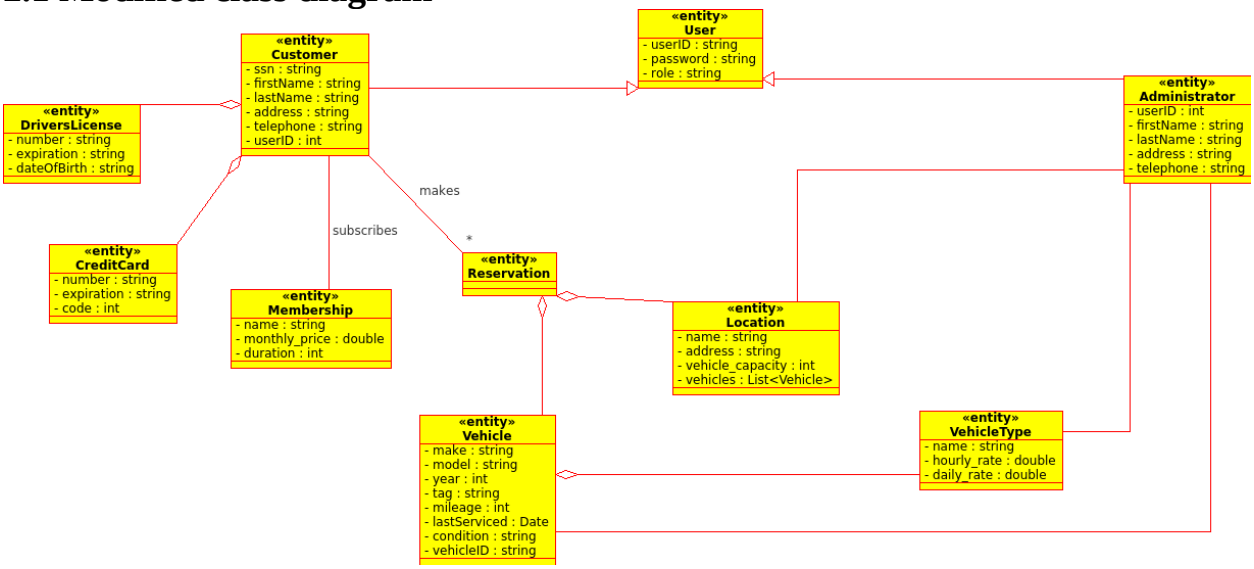
Vehicle type: a category of vehicle within the system where all vehicles in that categories share certain features (i.e. 7 seats), and share a common price. Examples: minivan, luxury car

1.4 References

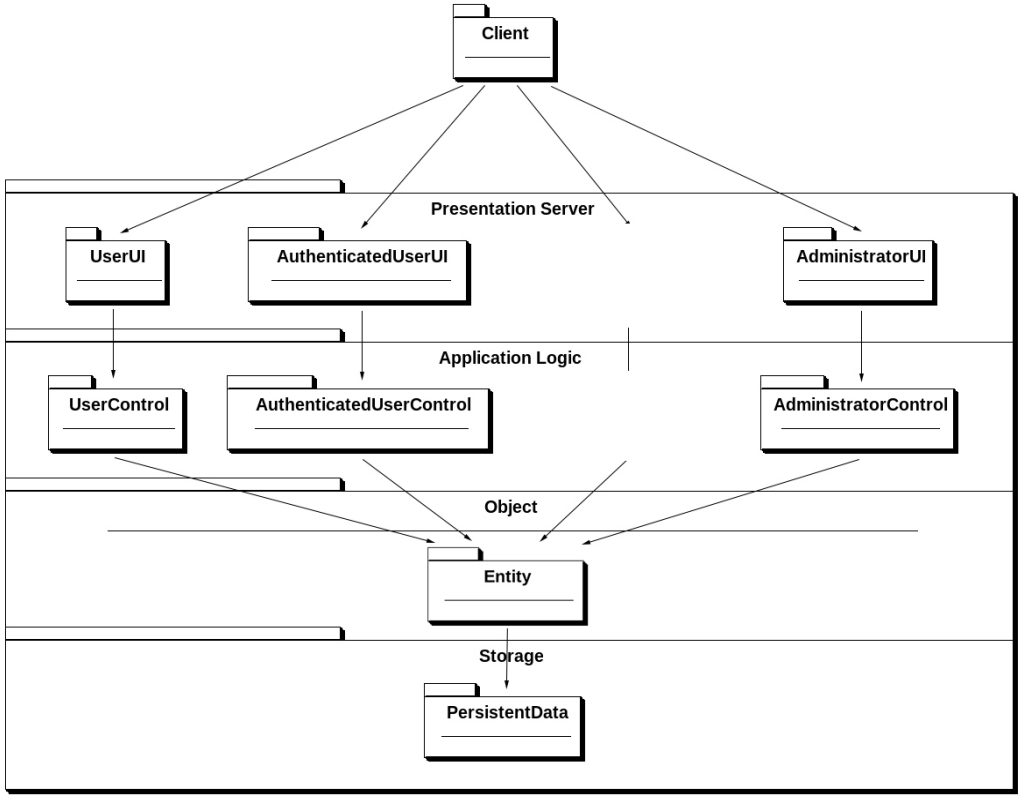
Java Enterprise Edition(Java EE) specifications

2. Packages

2.1 Modified class diagram



2.2 Packages, subsystems and layers



3. Class interfaces

3.1 Data dictionary

3.1.1 Entity Package

ClassName	User				
Purpose	The generalize entity involved in a Car rental system transaction				
SuperClass	None				
SubClass	Administrator, Customer				
Attributes	**Visibility**	**Name**	**Type**	**Description**	
	private	userID	Integer	The ID of the User as defined by the System Registration	
	private	password	String	The encryption on the user profile to be accessed	
	private	role	String	The role of the user defined by the system	
Operations	**Visibility**	**Name**	**Input**	**Output**	**Description**
	public	Login	userID password role		The login page allows users access to registered users within a system

	public	Logout			The logout page allows users currently login in to exit there current system profile
	public	updateProfile	profile		The update profile operation allows for user to change and save their profile information
Relationships	Super class of Administrator, Customer				
Constraints	**Context** User **inv**: userID <> nil and password.length >=6				
Exception	**login()** throws an **Exception** if **userID** and **password** do not match or no **userID** exists in the database				

ClassName	Customer				
Purpose	The generalization of group of Users labeled Customers				
SuperClass	Users				
SubClass	None				
Attributes	**Visibility**	**Name**	**Type**	**Description**	
	private	userID	Integer	The ID of the User as defined by the System Registration	
	private	ssn	String	Customer social security information for verification	
	private	firstName	String	Customer First name	
	private	lastName	String	Customer Last name	
	private	address	String	Customer Address	
	private	telephone	String	Customer Telephone number	
Operations	**Visibility**	**Name**	**Input**	**Output**	**Description**
	public	ViewMyInfo		Customer	Displays all personal information of the current logged in Customer
	public	ViewAvailableCars	carName carLocation	List<Vehicle>	Displays the Cars available for rental by location
	public	ViewMyRentalHistory		List<Vehicle>	Displays a current logged Customer rental history

	public	ViewMyBillingInfo		billingReport	Displays Current logged Customer billing report
	public	ViewMyCarRental		Vehicle	Displays Customer past and present car rentals
	public	EditMyInfo	userID firstName lastName address telephone	Customer	Allows Customers to edit their information as needed
Relationships	Subclass of the User A Customer can only rent 1 Car at a time A Customer can only edit personal information, but not vehicle information				
Constraints	**Context** Customer **inv**: ssn <> nil **Context** Customer **inv**: firstName <> nil **Context** Customer **inv**: lastName <> nil **Context** Customer **inv**: address <> nil **Context** Customer **inv**: telephone <> nil				
Exception	EditMyInfo() throws an Exception if the information input is null, or no modifications were made.				

ClassName	Administrator				
Purpose	The generalization of group of Users labeled Administrator within the Car rental System				
SuperClass	Users				
SubClass	None				
Attributes	**Visibility**	**Name**	**Type**	**Description**	
	private	userID	Integer	ID identifying the Administrator within the car rental system	
	private	firstName	String	Admin First name	
	private	lastName	String	Admin Last name	
	private	address	String	Admin Address	
	private	telephone	String	Admin Telephone number	
Operations	**Visibility**	**Name**	**Input**	**Output**	**Description**

	public	ViewMyInfo		Administrator	Displays all personal information of the current logged in admin
	public	ViewVehicles	CarName CarLocation	List<Vehicle>	Displays the Cars available within the Car rental system
	public	ViewRentalHistory	Customer Name	Customer	Displays a rental history for all customer by name
	public	ViewBillingHistory	Customer Name	Customer	Displays all Customer History Billing reports
	public	ViewCarRentals	Customer Name	Vehicle	Displays All Customer Car rental reports
	public	EditVehicleInfo	Vehicle Name	Vehicle	Edit Vehicle Information
	public	EditCustomerInfo	CustomerName	Customer	Allows Admin to edit customer information both adding and removing Customers
Relationships	Subclass of the User An Admin can edit Vehicle Information, and Customer information on Request				
Constraints	**Context** Administrator **inv**: userID <> nil **Context** Administrator **inv**: firstName <> nil **Context** Administrator **inv**: lastName <> nil **Context** Administrator **inv**: address <> nil **Context** Administrator **inv**: telephone <> nil				
Exception	None				

ClassName	DriversLicense
Purpose	Provide driver's license information of Customer
SuperClass	None

SubClass	None				
Attributes	**Visibility**	**Name**	**Type**	**Description**	
	private	number	String	The number on the license	
	private	expiration	String	The expiration date	
	private	dateOfBirth	String	The date of birth on the license	
Operations	**Visibility**	**Name**	**Input**	**Output**	**Description**
Relationships	None				
Constraints	**Context** DriversLicense **inv**: number <> nil **Context** DriversLicense **inv**: expiration <> nil **Context** DriversLicense **inv**: dateOfBirth <> nil				
Exception	None				

ClassName	CreditCard				
Purpose	Provide credit card information of Customer				
SuperClass	None				
SubClass	None				
Attributes	**Visibility**	**Name**	**Type**	**Description**	
	private	number	String	The number on the card	
	private	expiration	String	The expiration date	
	private	code	Integer	The CVC code	
Operations	**Visibility**	**Name**	**Input**	**Output**	**Description**
Relationships	None				
Constraints	**Context** CreditCard **inv**: number <> nil **Context** CreditCard **inv**: expiration <> nil **Context** CreditCard **inv**: code <> nil				
Exception	None				

ClassName	VehicleType				
Purpose	An entity class for storing information on vehicle types				
SuperClass	None				
SubClass	None				
Attributes	**Visibility**	**Name**	**Type**	**Description**	
	private	name	String	The name of the vehicle type	
	private	hourly_rate	Double	The hourly price of the vehicle type	
	private	daily_rate	Double	The daily price of the vehicle type	
Operations	**Visibility**	**Name**	**Input**	**Output**	**Description**
Relationships	None				

Constraints	**Context** VehicleType **inv**: name <> nil **Context** VehicleType **inv**: hourly_rate <> nil **Context** VehicleType **inv**: daily_rate <> nil
Exception	None

ClassName	Vehicle				
Purpose	An entity class for storing information on vehicles in the fleet				
SuperClass	None				
SubClass	None				
Attributes	**Visibility**	**Name**	**Type**	**Description**	
	private	vehicleID	String	The ID of the vehicles	
	private	make	String	The make of the vehicle	
	private	model	String	The model of the vehicle	
	private	year	Integer	The manufacture year of the vehicle	
	private	tag	String	The registration tag of the vehicle	
	private	mileage	Integer	The current mileage of the vehicle	
	private	last_serviced	Date	The last date the vehicle was serviced	
	private	condition	String	The condition of the vehicle	
Operations	**Visibility**	**Name**	**Input**	**Output**	**Description**
Relationships	A vehicle belongs to 1 location				
Constraints	**Context** Vehicle **inv**: vehicleID <> nil **Context** Vehicle **inv**: make <> nil **Context** Vehicle **inv**: model <> nil **Context** Vehicle **inv**: year <> nil **Context** Vehicle **inv**: tag <> nil **Context** Vehicle **inv**: mileage <> nil **Context** Vehicle **inv**: last_serviced <> nil **Context** Vehicle **inv**: condition <> nil				
Exception	None				

ClassName	Location			
Purpose	An entity class for storing information on point of presence POP locations			
SuperClass	None			
SubClass	None			
Attributes	**Visibility**	**Name**	**Type**	**Description**
	private	name	String	The name of the location
	private	address	String	The address of the location
	private	vehicle_capacity	Integer	The capacity of the location
	private	vehicles	List<Vehicle>	The list of vehicles assigned to the location

Operations	**Visibility**	**Name**	**Input**	**Output**	**Description**
Relationships	None				
Constraints	**Context** Location **inv**: name <> nil **Context** Location **inv**: address <> nil **Context** Location **inv**: vehicle_capacity <> nil **Context** Location **inv**: vehicles <> nil				
Exception	None				

ClassName	Membership			
Purpose	An entity class for storing information on membership tiers			
SuperClass	None			
SubClass	None			
Attributes	**Visibility**	**Name**	**Type**	**Description**
	private	name	String	The name of the membership tier
	private	monthly_price	Double	The price of one month's service
	private	duration	Integer	The length of membership in months

Operations	**Visibility**	**Name**	**Input**	**Output**	**Description**
Relationships	A customer belongs to 1 membership				
Constraints	**Context** Membership **inv**: name <> nil **Context** Membership **inv**: monthly_price <> nil **Context** Membership **inv**: duration <> nil				
Exception	None				

3.1.2 user.control package

ClassName	User				
Purpose	This control class allows a User to complete actions associated with login and profile modification				
SuperClass	None				
SubClass	None				
Attributes	**Visibility**	**Name**	**Type**	**Description**	
Operations	**Visibility**	**Name**	**Input**	**Output**	**Description**
	public	login	username password		Logs the user in with the appropriate permissions
	public	logout			Logs the user out
	public	loadProfile	username		Gets all relevant information related to the username

	public	updateProfile	profile	Boolean	Updates profile information and returns confirmation with Boolean
	public	registerCustomer	username password address creditcard driverLicense	Boolean	Allows a User to register as a customer
	public	vehicleSearch	name		Performs a search of vehicles in the fleet
Relationships	None				
Constraints	**Context** User::login(username, password) **pre**: username is not in the database **Context** User::login(username, password) **post**: username and password does not match				
Exception	updateProfile() throws **Exception** if fields are null or no information was modified.				

3.1.3 user.boundary package

ClassName	LoginUI				
Purpose	This boundary class allows a User to log in				
SuperClass	None				
SubClass	None				
Attributes	**Visibility**	**Name**	**Type**	**Description**	
Operations	**Visibility**	**Name**	**Input**	**Output**	**Description**
	public	login	username password		Verify the user and logs the user with appropriate permissions
Relationships	None				
Constraints	None				
Exception	None				

ClassName	LogoutUI				
Purpose	This boundary class allows a User to log out				
SuperClass	None				
SubClass	None				
Attributes	**Visibility**	**Name**	**Type**	**Description**	
Operations	**Visibility**	**Name**	**Input**	**Output**	**Description**
	public	logout			Logs the user out
Relationships	None				

Constraints	None
Exception	None

ClassName	ChangeProfileUI				
Purpose	This boundary class allows a User to perform profile changes				
SuperClass	None				
SubClass	None				
Attributes	**Visibility**	**Name**	**Type**	**Description**	
Operations	**Visibility**	**Name**	**Input**	**Output**	**Description**
	public	updateProfile	profile		Updates the profile of a user
	public	saveProfile	profile	Boolean	Saves changes of profile change to data structure
Relationships	None				
Constraints	None				
Exception	None				

ClassName	RegisterUI				
Purpose	This boundary class allows a User to register as a customer				
SuperClass	None				
SubClass	None				
Attributes	**Visibility**	**Name**	**Type**	**Description**	
Operations	**Visibility**	**Name**	**Input**	**Output**	**Description**
	public	registerCustomer	username password address creditcard driverlicense	Boolean	Registers a customer
Relationships	None				
Constraints	None				
Exception	None				

ClassName	SearchUI				
Purpose	This boundary class allows a User to search the vehicle fleet				
SuperClass	None				
SubClass	None				
Attributes	**Visibility**	**Name**	**Type**	**Description**	
Operations	**Visibility**	**Name**	**Input**	**Output**	**Description**

	public	search	keyword	List<Vehicle>	Gets a list of available vehicles based on keyword
Relationships	None				
Constraints	None				
Exception	None				

3.1.4 administrator.control package

ClassName	UserAdd				
Purpose	This control class allows Administrator to add users of all permissions				
SuperClass	None				
SubClass	None				
Attributes	**Visibility**	**Name**	**Type**	**Description**	
Operations	**Visibility**	**Name**	**Input**	**Output**	**Description**
	public	addUser	username password	Boolean	Adds a user to the database with Boolean confirmation
Relationships	None				
Constraints	None				
Exception	addUser() thows **Exception** with username or password is null				

ClassName	VehicleRemove				
Purpose	This control class allows Administrator to remove a vehicle from the system				
SuperClass	None				
SubClass	None				
Attributes	**Visibility**	**Name**	**Type**	**Description**	
Operations	**Visibility**	**Name**	**Input**	**Output**	**Description**
	public	removeVehicle	vehicleProfile	Boolean	Removes a vehicle from the fleet with Boolean conformation
Relationships	None				
Constraints	None				
Exception	None				

ClassName	VehicleTypeRemove			
Purpose	This control class allows an Administrator to remove a vehicle type from the system			
SuperClass	None			
SubClass	None			
Attributes	**Visibility**	**Name**	**Type**	**Description**

Operations	**Visibility**	**Name**	**Input**	**Output**	**Description**
	public	removeVehicleType	vehicleType	Boolean	Removes a vehicleType from the database with confirmation
Relationships	None				
Constraints	**Context** VehicleTypeRemove::removeVehicleType(vehicleType) **post**: vehicleType is not in the dataset				
Exception	removeVehicleType() throws an **Exception** if vehicleType is null				

3.1.5 administrator.boundary package

ClassName	UserAddUI				
Purpose	This boundary class allows an Administrator to add a user to the domain				
SuperClass	None				
SubClass	None				
Attributes	**Visibility**	**Name**	**Type**	**Description**	
Operations	**Visibility**	**Name**	**Input**	**Output**	**Description**
	public	addUser	username password	Boolean	Adds a user to the system confirmation
Relationships	None				
Constraints	None				
Exception	None				

ClassName	RemoveVehicleUI				
Purpose	This boundary class allows Administrator to remove a vehicle from the fleet				
SuperClass	None				
SubClass	None				
Attributes	**Visibility**	**Name**	**Type**	**Description**	
Operations	**Visibility**	**Name**	**Input**	**Output**	**Description**
	public	removeVehicle	vehicleProfile	Boolean	Removes a vehicle from the fleet with conformation
Relationships	None				
Constraints	None				
Exception	None				

ClassName	RemoveVehicleTypeUI
Purpose	This boundary class allows Administrator to remove a vehicleType from the system

SuperClass	None				
SubClass	None				
Attributes	**Visibility**	**Name**	**Type**	**Description**	
Operations	**Visibility**	**Name**	**Input**	**Output**	**Description**
	public	removeVehicleType	vehicleType	Boolean	Removes a vehicleType from the site with confirmation
Relationships	None				
Constraints	None				
Exception	None				

3.1.6 customer.control package

ClassName	Membership				
Purpose	This control class allows a Customer to perform actions to their membership options				
SuperClass	None				
SubClass	None				
Attributes	**Visibility**	**Name**	**Type**	**Description**	
Operations	**Visibility**	**Name**	**Input**	**Output**	**Description**
	public	terminateMembership		Boolean	Terminates a membership of a customer with Boolean confirmation
Relationships	None				
Constraints	**Context MembershipCtrl::** terminateMemberShip(): Termination will be triggered by a button within the CustomerUI boundary				
Exception	None				

ClassName	VehicleReservation				
Purpose	This control class allows a Customer to preform actions related to renting a vehicle				
SuperClass	None				
SubClass	None				
Attributes	**Visibility**	**Name**	**Type**	**Description**	
Operations	**Visibility**	**Name**	**Input**	**Output**	**Description**
	public	reserveVehicle	vehicleProfile	Boolean	Reserves a vehicle with Boolean confirmation

	public	returnVehicle	vehicleProfile	Boolean	Checks a vehicle back into the fleet of available vehicles with Boolean confirmation
Relationships	None				
Constraints	**Context ReservationCtrl::** reservation(vehicleProfile) pre: vehicleProfile is in the data set **Context ReservationCtrl::** returnVehicle(vehicleProile) pre: vehicleProfile is contained within the dataset				
Exception	None				

3.1.7 customer.boundary package

ClassName	MembershipUI				
Purpose	This boundary class allows a Customer to terminate their membership				
SuperClass	None				
SubClass	None				
Attributes	**Visibility**	**Name**	**Type**	**Description**	
Operations	**Visibility**	**Name**	**Input**	**Output**	**Description**
	public	terminateMembership		Boolean	Terminates a customer's membership with conformation
Relationships	None				
Constraints	None				
Exception	None				

ClassName	ReservationUI				
Purpose	This boundary class allows a Customer to reserve a vehicle				
SuperClass	None				
SubClass	None				
Attributes	**Visibility**	**Name**	**Type**	**Description**	
Operations	**Visibility**	**Name**	**Input**	**Output**	**Description**
	public	reserveVehicle	vehicleProfile	Boolean	Reserves a vehicle with Boolean confirmation
	public	returnVehicle	vehicleProfile	Boolean	Checks a vehicle back into the fleet of available vehicles with Boolean confirmation
Relationships	None				
Constraints	None				
Exception	None				

3.2 Factory methods for the persistent and object layers

3.2.1 entity

package uga.cs.x050.team4.entiy;
public interface EntityFactory {
public Customer createCustomer(Int userID, String ssn, String firstName, String lastName, String address, String telephone) throws CarRentalException;
public Administrator create Administrator(Int userID, String firstName, String lastName, String address, String telephone) throws CarRentalException;
}

3.2.2 Persistent

package uga.cs.x050.team4.persistent;
public interface EntityPersistentFactory {
public Customer storeCustomer(Int userID, String ssn, String firstName, String lastName, String address, String telephone) throws CarRentalException;
public Customer restoreCustomer(Int userID) throws CarRentalException;
public Iterator restoreCustomer() throws CarRentalException;
public Administrator storeAdministrator(Int userID, String firstName, String lastName, String address, String telephone) throws CarRentalException;
public Administrator restoreAdministrator(Int userID) throws CarRentalException;
public Iterator restoreAdministrator() throws CarRentalException;
public DriversLicense storeDriversLicense(String number, String expiration, String dateOfBirth) throws CarRentalException;
public DriversLicense restoreDriversLicense(String number) throws CarRentalException;
public Iterator restoreDriversLicense() throws CarRentalException;
public CreditCard storeCreditCard(String number, String expiration, Int code) throws CarRentalException;
public CreditCard restoreCreditCard(String number) throws CarRentalException;
public Iterator restoreCreditCard() throws CarRentalException;
public VehicleType storeVehicleType(String name, Double hourly_rate, Double daily_rate) throws CarRentalException;

public VehicleType restoreVehicleType(String name) throws CarRentalException;
public Iterator restoreVehicleType() throws CarRentalException;
public Vehicle storeVehicle(String vehicleID, String make, String model, Int year, String tag, Int mileage, Date last_serviced, String condition) throws CarRentalException;
public Vehicle restoreVehicle(String vehicleID) throws CarRentalException;
public Iterator restoreVehicle() throws CarRentalException;
public Location storeLocation(String name, String address, Int vehicle_capacity, List<Vehicle> vehicles) throws CarRentalException;
public Location restoreLocation(String name) throws CarRentalException;
public Iterator restoreLocation() throws CarRentalException;
public Membership storeMembership(String name, Double monthly_price, Int duration) throws CarRentalException;
public Membership restoreMembership(String name) throws CarRentalException;
public Iterator restoreMembership() throws CarRentalException;
}

3.2.3 Association

```
package uga.cs.x050.team4.associations;
public interface aggreagationModel {
        public aggreagationModel(long id) throws CarRentalException;
        public Iterator restoreAggreagationModels(Entity E) throws CarRentalException;
        public long storeAggregationModel(AggregationModel m) throws CarRentalException;
};

public interface reservationBy {
         public reservationBy     restoreReservationBy(long id) throws CarRentalException;
        public Iterator            restoreReservationBy(Customer C) throws CarRentalException;
       public  Iterator           restoreReservationBy(Vehcile V) throws CarRentalException;
      public long                storeReservationBy(reservationBy) throws CarRentalException;
};

public interface administratorMaintainsCustomer extends aggregationModel {
        public aggreagationModel(long id) throws CarRentalException;
        public Iterator restoreAdministratorMaintainsCustomer(Customer C) throws
        CarRentalException;

      public AdministratorMaintainsCustomer
restoreAdministratorMaintainsCustomer(Customer C)     throws CarRentalException;

        public long storeAggregationModel(CustomerProfile m) throws CarRentalException;

};

public interface adminMaintainsCustomer extends aggregationModel {
```

```
        public aggreagationModel(long id) throws CarRentalException;
       public Iterator restoreAdministratorMaintainsCustomer(Customer C) throws
CarRentalException;
        public AdministratorMaintainsCustomer
restoreAdministratorMaintainsCustomer(Customer E)    throws CarRentalException;

        public long storeAggregationModel(CustomerProfile m) throws CarRentalException;

};


public interface Locations extends aggregationModel {
        public aggreagationModel(long id) throws CarRentalException;
         public Iterator Location restoreLocation(Location L ) throws CarRentalException;

         public Location restoreAdministratorMaintainsCustomer(Location L)  throws
        CarRentalException;

        public long storeAggregationModel(Location L) throws CarRentalException;

};

public interface maintainVehicles {
        public  maintainVehicle   restoreMaintainVehicle(long id) throws CarRentalException;
        public Iterator            restoreMaintainVehicle (Vehicle V) throwCarRentalException;
       public  Iterator           restoreMaintainVehicle (VehcileInfo D) throws
CarRentalException;
      public long                storeReservationBy(maintainVehicles) throws
CarRentalException;
};

public interface maintainCustomerInfo {
        public  maintainVehicle   restoreMaintainVehicle(long id) throws CarRentalException;
        public Iterator            restoreMaintainVehicle (Customer C) throwCarRentalException;
       public  Iterator           restoreMaintainVehicle (CustomerInfo D) throws
CarRentalException;
      public long                storeReservationBy(maintainCustomerInfo) throws
CarRentalException;
};
```

4. Glossary

Abstract Factory pattern: provides an abstract class for each object that can be substituted and provides an interface for creating groups of objects

Adapter pattern: provides a different interface to en existing component, used to convert the interface of an existing piece of code into an interface that a calling subsystem expects

Analysis object model: describes the entity, boundary, and control objects that are visible to the user

Boundary use cases: describe, from the user's point of view, administrative and exceptional cases that the system handles

Contracts: constraints on a class that enable class users, implementers, and extenders to share the same assumption about the class. Contracts include three types of constraints: invariant, precondition, and postcondition.

Delegation: the alternative to implementation inheritance that should be used when reuse is desired.

Design pattern: a template solution that developers have refined over time to solve a range of recurring problems. A design pattern includes a name, a problem description, a solution, and a set of consequences. There are three types of patterns: creational, structural, and behavioral patterns

Development Cost: cost of developing the initial system.

Façade pattern: allows developers to further reduce dependencies between classes by encapsulating a subsystem with a simple, unified interface

Invariant: a predicate that is always true for all instances of a class. Invariants are constraints associated with classes or interface. Invariants are used to specify consistency constraints among class attributes

Object Constraint Language (OCL): a formal language defined as part of the UML used for expressing constraint

Object Design Document (ODD): a document describing the object design model. The object design model is often generated from comments embedded in the source code. There are three main approached to document object design: Self-contained ODD generate from model, ODD as extension of the RAD, and ODD embedded into source code

Object model restructuring: transform the object design model to improve its understandability and extensibility.

Object model optimization: transform the object design model to address performance criteria such as response time or memory utilization.

Postcondition: a predicate that must be true after an operation is invoked, used t specify constraints that the class implementor and the class extender must ensure after the invocation of the operation

Precondition: a predicate that must be true before an operation is invoked associated with a specific operation. Preconditions are used t specify constraints that a class user must meet before calling the operation

Reuse: indentify off-the shelf components and design pattern to make use of existing solution.

Scalability: a system’s ability to process more workload, with a proportional increase in system resource usage.

Service specification: describe each class interface.

Simplicity: a system should be well-designed, system, and tools are usually reliable, easy to use and maintain, and simple in concept.

Singleton pattern: ensure a class only has one instance, and provide a global point of access to it

System Design Document: Car Rental Service

Group 4
CSCI 4050: Software Engineering
Instructor: Krys Kochut
Osama Mansour, Stephen Patton, Vincent Lee, and Minh Pham

Table of Contents

1. Introduction

1.1 Purpose of the system

The purpose of the system is to support an online car rental service for young drivers. Customers can sign up, log in, search for cars, reserve cars, return cars, and pay fees. The system is designed to have near continuous uptime and support every conceived need the customer will have.

1.2 Design goals

Ease of use. We want our customers to have no difficulties understanding our website. The design will be implemented in such a way that potential customers can easily use the service from the beginning.

High availability. The more customers we have registered with the system, the more potential rentals the system can handle. It is valuable to have the system ready to handle many concurrent users and to do so with minimal downtime.

Scalability. As the company grows, so will the databases. The implementation design now should be scalable to accommodate a growing user base.

Ease of adding new vehicles and locations. The system is designed in hopes that the business will grow. With that in mind, it must be simple, expedient, and secure to add new locations, vehicles, and vehicle type categories. Full time administrators will be hired to manage these.

1.3 Definitions

Vehicle type: a category of vehicle within the system where all vehicles in that categories share certain features (i.e. 7 seats), and share a common price. Examples: minivan, luxury car

1.4 Reference

Document No.	Document Title	Date	Author
1	Object-Oriented Software Engineering, Using UML, pattern and Java	2010	Bernd Bruegge, Allen H.Dutoit
2	Operating System Concepts with Java	2010	Silberschatz, Galvin, Gagne
3	Database System, An Application-Oriented Approach	2006	Kifer, Bersntein, Lewis

HyperText Markup Language (HTML): the main markup language for creating web pages and other information that can be displayed in a web browser
JavaBeans: reusable software components for Java – classes that encapsulate many objects into a single object (the bean)
JavaScript (JS): an interpreted computer programming language which implementations allow client-side scripts to interact with the user, control the browser, communicate asynchronously, and alter the document content that is displayed

JavaServer Pages (JSP): a technology that helps software developers create dynamically generated web pages based on HTML, XML, or other document type
MySQL: open source, multiuser database management system known for its reliability and efficiency

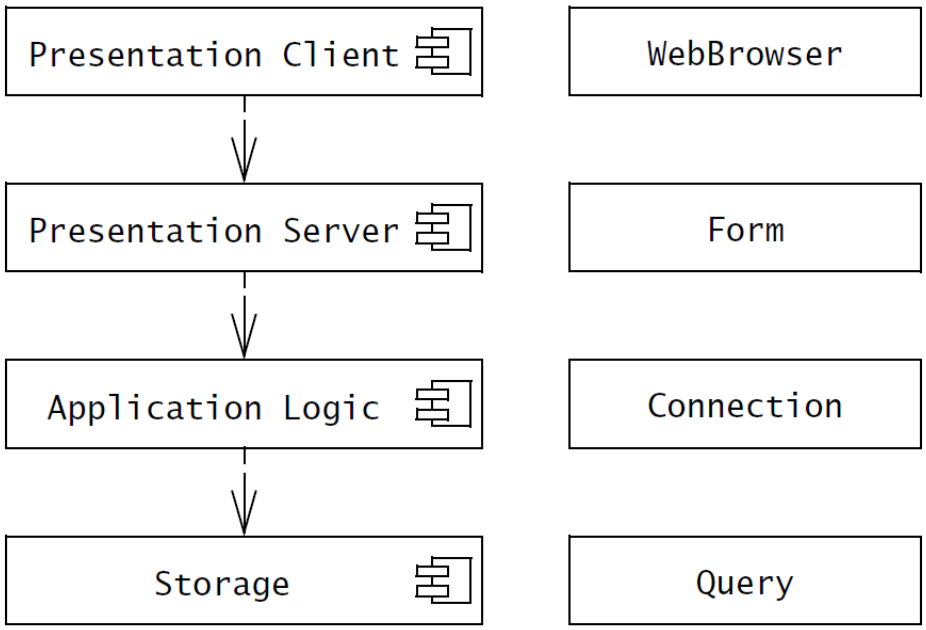
1.5 Overview

This document has been developed by Group 4 for the Car Rental Service Project. This document was developed from the above resources and is intended to satisfy all the customer requirements, objectives and expectations. This document also covers the current system architecture and our proposed architecture and subsystem service
The proposed architecture section details the subsystems of our implementation. It will provide information about how the system will handle subsystem decomposition, hardware/software mapping, persistent data, access control and security, global software control, and boundary conditions.
The subsystem services section proposes a basis for interface creation. It helps clarify boundaries among subsystems.

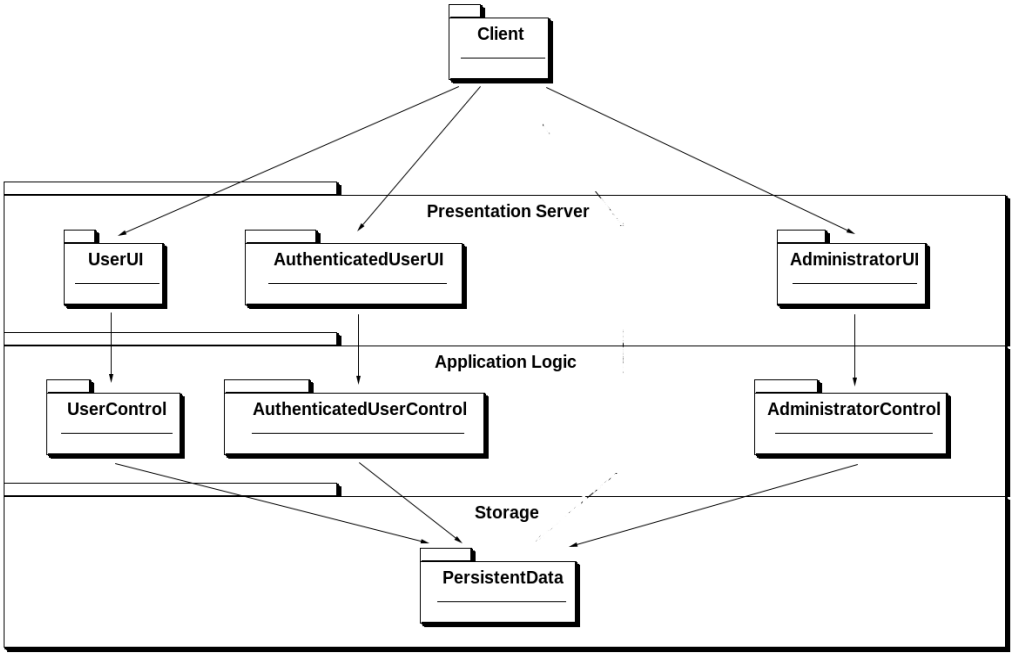
2. Proposed software architecture

2.1 Overview

The car rental service, You Drive, consists of a four-tier architectural style. This style consists of the presentation client, presentation server, application logic, and storage layers. The presentation client layer is located on the user's machines, while the presentation server layer is located on a cluster of servers. The application logic layer includes all control and entity objects, and the storage layer realizes the storage, retrieval, and query of persistent objects.



2.2 Subsystem decomposition



2.2.1 Presentation Layer – UserUI Subsystem

Operation	Returns	Arguments(s)	Description
logIn		username password	Verifies the user identity and log the user in or display message.
register		username password socialSecurityNumber firstName lastName address telephone creditCard	Registers a AuthenticatedUser with the given credentials and information or displays an error message.
locationSearch	List<Location>	keyword	Searches the available locations in the system.
vehicleSearch	List<Vehicle>	keyword	Searches the vehicles in the system.

2.2.2 Presentation Layer – AuthenticatedUserUI Subsystem

Operation	Returns	Arguments(s)	Description
logout			Log the user out.
ViewMyInfo	Customer		Displays all personal information of the current logged in Customer.
ViewAvailbileCars	List<Vehicle>	carName carLocation	Displays the Cars available for rental by location.
ViewMyRentalHistory	List<RentalHistory>		Displays a current logged Customer rental history.
ViewMyBillingInfo	Customer		Displays Current logged Customer billing report.
ViewMyCarRental	CarRental		Displays Customer past and present car rentals.
EditMyInfo	Customer	firstName lastName address telephoneNumber	Allows Customers to edit their information as needed.

2.2.3 Presentation Layer – AdministratorUI Subsystem

Operation	Returns	Arguments(s)	Description
logout			Log the user out.
ViewMyInfo	Administrator		Displays all personal information of the current logged in admin.
ViewVehicles	List<Vehicle>	carName carLocation	Displays the Cars available within the Car rental system.
ViewRentalHistory	List<RentalHistory>	customerName	Displays a rental history for all customer by name.
ViewBillingHistory	List<BillingHistory>	customerName	Displays all Customer History Billing reports.
ViewCarRentals	List<CarRental>	customerName	Displays All Customer Car rental reports.
EditVehicleInfo	Vehicle	make model tag mileage last_serviced condition	Edit Vehicle Information.
EditCustomerInfo	Customer	socialSecurityNumber firstName lastName address telephone	Allows Admin to edit customer information both adding and removing Customers.

2.2.4 Application Logic Layer – UserControl Subsystem

Operation	Returns	Arguments(s)	Description
logIn	boolean	username password	Verifies the user identity and log the user in or display message.
register	boolean	username password socialSecurityNumber firstName lastName address telephone creditCard driverLicense	Registers an AuthenticatedUser with the given credentials and information or displays an error message.

locationSearch	List<Location>	keyword	Searches the available locations in the system.
vehicleSearch	List<Vehicle>	keyword	Searches the vehicles in the system.

2.2.5 Application Logic Layer – AuthenticatedUserControl Subsystem

Operation	Returns	Arguments(s)	Description
logOut			Log the user out.
GetMyInfo	Customer		Returns customer personal information of the current logged in Customer.
GetAvailbileCars	List<Vehicle>	carName carLocation	Returns the Cars available for rental by location.
GetMyRentalHistory	List<RentalHistory>		Returns a current logged Customer rental history.
GetMyBillingInfo	Customer		Returns Current logged Customer billing report.
GetMyCarRental	CarRental		Returns Customer past and present car rentals.
GetMyInfo	Customer	firstName lastName address telephoneNumber	Returns Customer information.

2.2.6 Application Logic Layer – AdministratorControl Subsystem

Operation	Returns	Arguments(s)	Description
logOut			Log the user out.
GetMyInfo	Administrator		Returns all personal information of the current logged in admin.
GetVehicles	List<Vehicle>	carName carLocation	Returns the Cars available within the Car rental system.
GetRentalHistory	List<RentalHistory>	customerName	Returns a rental history for all customer by name.
GetBillingHistory	List<BillingHistory>	customerName	Returns all Customer History Billing reports.

GetCarRentals	List<CarRental>	customerName	Returns All Customer Car rental reports.
GetVehicleInfo	Vehicle	make model tag mileage last_serviced condition	Return Vehicle Information.
GetCustomerInfo	Customer	socialSecurityNumber firstName lastName address telephone	Returns customer information for editing.

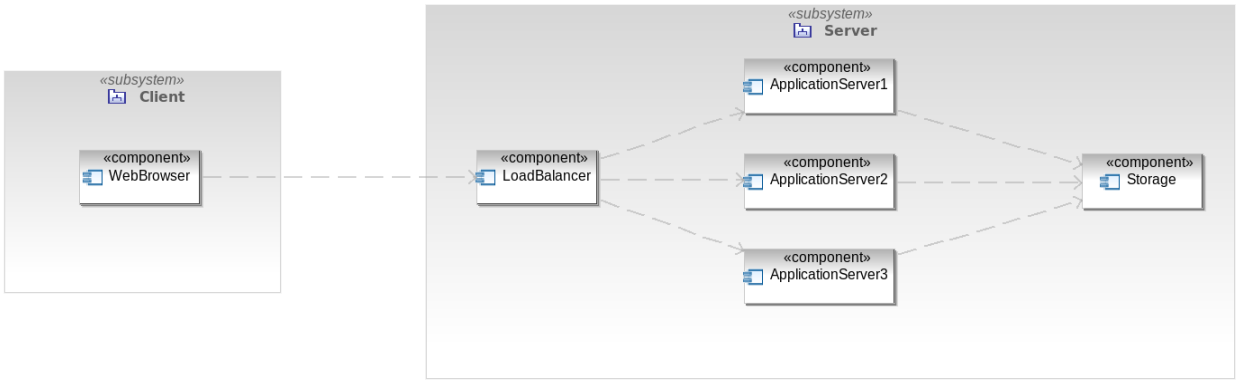
2.3 Hardware/software mapping

The four-tier architectural style is realized in hardware as two subsystems: Client and the Server. The Client subsystem contains the Presentation Client while the Server subsystem contains the Presentation Server, Application Logic, and Storage. An additional component, the Load Balancer distributes network traffic across a number of servers in the subsystem. In overview, the Client connects with a Load Balancer which routes the user's request to one of the three Application Server's which then communicates with the centralized storage to finish processing the request.

The presentation client is a web browser which is a software application for retrieving, presenting and traversing information resources from the server subsystem. The web browser also is capable of executing JavaScript (JS) which is a client scripting language that communicate asynchronously, and can alter the document content that is displayed in the web browser.

The presentation server and application logic layers reside on one of the three Application Server's. The interface is realized with HTML styled with CSS. The HTML is wrapped in JavaServer Pages (JSP) which allows dynamically generated web page creation. The JSP pages use the Java programming language, which is deployed on a servlet container Tomcat. Inside the Tomcat server, are JavaBean's which contain the application logic or connection required to communicate with the storage system.

The storage system consists of a MySQL Server relational database management system (RDBMS). This persistent storage is accessed by one of the three application servers providing abstraction from the system ensuring security, and data integrity.



2.4 Persistent Data Management

We will be using the MySQL platform to develop our databases. MySQL contains a relational relationship between objects and ID associated with them which allows us to access information for modification, querying, and deletion. Below we have list the persistent objects and explained important attributes that are accessed and their interactions within the car rental system.

2.4.1 Persistent Objects

- **CustomerUser**: contains the attributes identifying a customer within the Car Rental System
    - Is Referenced by **BillingInfo**, which contains relating information needed to view user billing
    - Is Reference by **RentalHistory**, which contains relating information needed to display a historical account of user car rentals
- **Administrator:** contains information relating the an administrator within the Car Rental System
    - The Administrator can add/delete/modify all and any types of information within the database system within including all persistent objects

2.4.2 Storage Strategy

We will be using the MySQL relations database for our storage needs with the Car rental System.

- MySQL is a free, accessible relational database system that has little barriers in configuration
- MySQL is lightweight, and quick for querying and modifying databases with in an optimized manner
- MySQL is also widely used database with basic standards established for many types of Qualified Database Administrators, Analyst, and Designers.

2.5 Access and Control Security

	User	Cars	Rental	Billing Info	Rental History
User (Non-Authenticated)	none	search	none	none	none
CustomerUser	search rent	search	rent	View	View
AdministrativeUser	Create Update Delete Search	Create Update Delete Search	View Update	View Update	View Update

	Car Type	Customer Rentals	Admin Info	User Defined Attribute Names	User Defined Attribute Values
User(Non-Authenticated)	none	none	none	none	none
Customer	none	none	none	saved	saved
Administrator	Create Delete	View Update	View Create Update Delete	saved	saved

2.6 Global Software Control

The system will be implemented with Servlets and JSP approach, the *procedural driven control* is ideal for maintaining a simple path oriented way of controlling data flow within the server side of the system. Rental Transactions and Threaded processes in implementation will also follow procedural driven logic processes.

An Example of Client/Server Interaction:

The available server will wait for a request via HTTP to the servlet based inside the web server, and on execution of the procedural request a thread will be created with the following constraints

- If any number of rental transactions are being performed then availability statuses need to be updated in order to reduce concurrent rental of non-existent vehicles.
- If transactions are successful, the user should receive a message pertaining to their request be it a rental, billing information, car information request, object manipulation, or other system components.
- If transaction fails, system should revert back to old state to avoid irreversible data. The user shall also receive an appropriate message pertaining to failure.

All Transactions request should uphold this follow of client/server interactions within the system. All user who have made a request and succeed to a rental transaction page should be given their request with 100% accuracy. The system will perform an inventory check before allowing the user into the rental transaction page if not a message will occur notifying the user. System is to minimize any deadlock or race conditions occurring during concurrent transactions.

2.7 Boundary Conditions

2.7.1 Start-up conditions

The System administrator must adhere to the following to execute a correct system startup.

1. The System will begin by starting MySQL Database
2. The System will start the Apache Tomcat web server with Servlet Containers

2.7.2 Shut-down conditions

To prevent partial writes to the database, this shutdown procedure must be followed.

1. The System will shut down by closing the Apache Tomcat Webserver
2. The System will shut down by closing the MySQL Database

2.7.3 System-Failure conditions

Exceptions might occur during the operation of the system, messages are used to announce their presence.

1. *Client-Side Exception:* If invalid data is input to the system with the presentation layer of the system. An Error message is displayed to the client with no system changes edits.
2. *Server-Side Exceptions:* All errors occurring on the server side shall first proceed with a message to the current user with diagnosis options. Dependent upon the users preference the Server-side will execute the following desired process.(Ex: Database Rollbacks or Information restorations)

3. Glossary

Apache Tomcat (Tomcat): an open source web server and servlet container developed by the Apache Software Foundation (ASF). It implements the Java Servlet and the JavaServer Pages specifications from Sun Microsystems, and provides a pure Java HTTP web server environment for Java code to run in.

Application layer: a layer in architectural style that includes all control and entity objects, realizing the processing, rule checking, and notification required by the application.

Attribute: a name paired with a domain (commonly referred to as type or data type).

Cascading Style Sheets (CSS): a style sheet language used for describing the presentation semantics of a document written in a markup language. The most common application is to style web pages written in HTML and XHTML.

Client/Server Architecture: an architectural style in which a subsystem, the server, provides service to instance of the other subsystems called the clients, which are responsible for interacting with the user.

Concurrency: a property of system in which several transactions are executing simultaneously, possibly interacting with each other.

Data integrity: maintaining and assuring the accuracy and consistency of data over its entire life cycle, and is a critical aspect to the design, implementation and usage of any system which stores, processes or retrieves data.

Deadlock: a situation in which two or more competing actions are each waiting for the other to finish, and thus neither ever does. Deadlock can happen in transactional database and operating system.

Four-tier architecture: an architectural style that organizes subsystem into four layers: the interface layers which decomposed into a Presentation Client layer and Presentation Server, the application logic layer, and the storage layer.

Global software control: describes how subsystems synchronize, including listing and addressing synchronization and concurrency issue.

Hardware Software Mapping: a process to realize the subsystems by mapping the objects and subsystem onto the chosen hardware and/or software. This process includes two types: mapping objects and mapping associations.

Interface layer: a layer in architectural style that includes all boundary objects that deal with the user, including windows, forms, web pages, and so on.

Load Balancer: a computer networking software or hardware for distributing workloads across multiple computing resources, such as computers, a computer clusters, network cluster, network links, central processing units or disk drives.

Persistent data: data which has been specifically set by the user. It is never destroyed even when a parameter is hooked up to a source object.

Persistent data management: control the access to persistent data through the menu.

Persistent object: a unique identifier of a record on a table, used as the primary key.

Procedural driven control: ideal for maintaining a simple path oriented of controlling data flow within the server side of the system.

Race condition: where several transactions access and manipulate the same data concurrently and the outcome of the execution on the particular order in which the access takes place, is called a race condition.

Relation: a set of tuples (d_1, d_2, …, d_n) where each element d_i is a member of D_i, a data domain.

Relational database: a finite set of relation, including a set of relation schemas-called database schema, and a set of corresponding relation instances-called database instance.

Relational database management system (RDBMS): a database management system (DBMS) that is based on the relational model.

SQL: Structured Query Language designed for managing data held in a relational database management system.

Storage layer: a layer in architectural style that realizes the storage, retrieval, and query of persistent objects.

Subsystem decomposition: identification of subsystems, service, and their association to each other (hierarchical, peer-to-peer, etc.).

Transaction: the change of the state of the enterprise corresponding to the change of the information stored in the database in real time by program.

**subsystem services not included on this iteration of the SDD*

YouDrive Metrics

of Packages: 5 Packages
# of Classes: 55 Classes
# of Lines of Code Total: 12,325 Lines of Code
# of Ant Files: 1 Ant Build File
# of JSP Pages: 45 pages