RU Parking

ANDROID-BASED APPLICATION FOR SUGGESTING PARKING SPACES

http://code.google.com/p/ruparking/

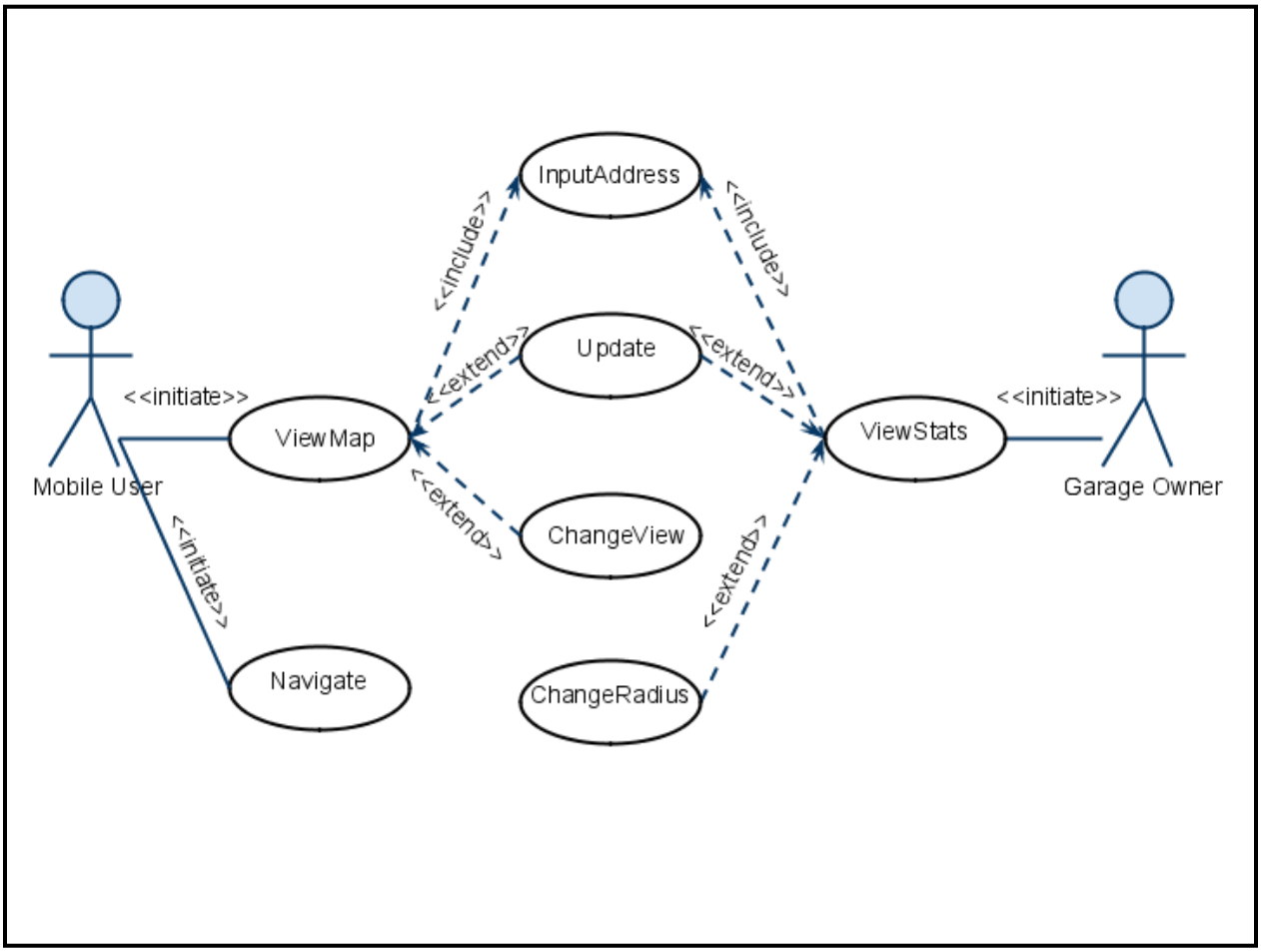
OBJECT-ORIENTED ANALYSIS REPORT

Project Members:

Cyrus Gerami
Ashwin Revo
Shweta Sagari
Samson Sequeira
Vrajesh Vyas

Contents

1. PRIORITIZATION OF USE CASES



From the above use case diagram we will implement the use cases in the following order of priority:

1. ViewMap
2. ViewStats
3. Navigate
4. ChangeView
5. ChangeRadius
6. Update

2. TEXTUAL ANALYSIS OF REQUIREMENTS ANALYSIS REPORT

2.1 Textual analysis

The Use cases in the requirements analysis document were subjected to a textual analysis to identify the following objects:

Use case Name	ViewMap
Participating Actors	Initiated by MobileUser
Flow of Events	1. MobileUser presses the "OpenApplication" button which creates an "InputAddress" form. 2. In this form the user can type in the destination address or select the current location as his input. Once the form is completed the MobileUser submits the form by pressing the "Submit" button, at which point the Server is notified. 3. The Server parses the information submitted by the MobileUser and submits parking availability data to the Application. 4. The Application uses this data to create a map which is then displayed on the LCD.
Entry Condition	MobileUser activates the "ViewMap" function of his phone.
Exit Condition	• The MobileUser presses the "ExitApplication" button. • The MobileUser presses the "ChangeView" button. • The MobileUser presses the "SelectStreet" button. • The MobileUser presses the "Update" button.
Quality Requirements	• Parking suggestion should be in real-time and accurate. • The colors in the map should be easily identifiable.

Figure 2-1 The ViewMap use case description.

Use case Name	ViewStats
Participating Actors	Initiated by GarageOwner
Flow of Events	1. GarageOwner presses the “OpenApplication” button which creates an “InputAddress” form. 2. In this form the user can type in the destination address or select the current location as his input. Once the form is completed the GarageOwner submits the form by pressing the “Submit” button, at which point the Server is notified. 3. The Server parses the information submitted by the GarageOwner and submits parking availability data to the Application. 4. The Application uses this data to create a list which is then displayed on the LCD.
Entry Condition	GarageOwner activates the “ViewStats” function of his phone.
Exit Condition	• The GarageOwner presses the “ExitApplication” button. • The GarageOwner presses the “ChangeRadius” button. • The GarageOwner presses the “Update” button.
Quality Requirements	1. Parking suggestion should be in real-time and accurate.

Figure 2-2 The ViewStatistics use case description.

Use case Name	Navigate
Participating Actors	Initiated by Mobile User
Flow of Events	1. The MobileUser presses the “SelectStreet” button. 2. Now the MobileUser is asked to confirm this by pressing a “Navigate” button. 3. A navigate request is sent to the Application. 4. The Application calls the “AddRouteToMap” function which generates the routing information. 5. This information is then sent to the LCD to be displayed.
Entry Condition	The MobileUser uses the “ViewMap” function.
Exit Condition	• The MobileUser presses the “ExitApplication” button. • The MobileUser pressing the “Cancel” button when asked for confirmation.
Quality Requirements	• The Application should suggest the best route on the basis of shorter distance or lesser time.

	• The directions should be timed correctly so that the user is informed on time.

Figure 2-3 The Navigate use case description.

Use case Name	InputAddress
Participating Actors	Initiated by MobileUser or GarageOwner
Flow of Events	1. MobileUser or GarageOwner presses the "OpenApplication" button which creates an "InputAddress" form. 2. In this form the user can type in the destination address or select the current location as his input. Once the form is completed the MobileUser or GarageOwner submits the form by pressing the "Submit" button, at which point the Server is notified.
Entry Condition	The MobileUser or the GarageOwner activates the Application.
Exit Condition	• The MobileUser or GarageOwner clicks on the "ExitApplication" button. • The MobileUser or GarageOwner clicks on the "Submit" button.
Quality Requirements	The form should be able to differentiate between street names, state names and ZIP codes.

Figure 2-4 The InputAddress use case description.

Use case Name	ChangeView
Participating Actors	Initiated by MobileUser
Flow of Events	1. The MobileUser presses the"ChangeView" button while viewing the map on the LCD. 2. This sends a "ChangeView" request to the Application. 3. The Application then creates a new map having a different detail level. 4. This information is then sent to be displayed on the LCD.
Entry Condition	The MobileUser uses the "ViewMap" function.
Exit Condition	The Application displays the new map on the LCD.
Quality Requirements	The "ChangeView" function should have 3 distinct levels of detail to provide varying amounts of information to the user.

Figure 2-5 The ChangeView use case description.

Use case Name	ChangeRadius
Participating Actors	Initiated by GarageOwner
Flow of Events	1. The GarageOwner presses the”ChangeRadius” button while viewing the list on the LCD. 2. This sends a “ChangeRadius” request to the Application. 3. The Application then creates a new list having a different amount of information. 4. This information is then sent to be displayed on the LCD.
Entry Condition	The GarageOwner uses the “ViewStats” function.
Exit Condition	The Application displays the new list on the LCD.
Quality Requirements	The “ChangeRadius” function should have 3 distinct details of information to provide to the user.

Figure 2-6 The ChangeRadius use case description.

Use case Name	Update
Participating Actors	Initiated by Mobile User and Garage Owner
Flow of Events	1. The MobileUser or GarageOwner presses the “Update” button while viewing the map or list. 2. The request for update is sent to the Application. 3. The Application then submits the address to the Server. 4. The Server parses this information and sends relevant parking availability information to the Application. 5. The Application then creates a new map or list which is displayed on the LCD.
Entry Condition	The MobileUser uses the “ViewMap” function.
Exit Condition	The Application displays the new map or list on the LCD.
Quality Requirements	While updating the previous destination address of the user should be used.

Figure 2-7 The Update use case description.

2.2 Identifying Entity Objects

MobileUser	A person who uses the application to find vacant parking spaces. A MobileUser starts the application, inputs the address and views the map displaying parking information. MobileUsers are identified by their name.
GarageOwner	A person who uses the application to find parking information around his parking garage. A GarageOwner starts the application, inputs the address and views the statistics of the parking information. GarageOwners are identified by their name.
Server	Server has data regarding vacant parking spaces. The Application interacts with the Server and requests parking information relevant to a specific user's query. The Server returns the parking space availability around the requested location.

2.3 Identifying Boundary Objects

OpenAppButton	Button used by the MobileUser and the GarageOwner to activate the application.
InputAddressForm	Form used by the application to input the address. This form is presented to the User when the application is activated. The InputAddressForm contains a button to submit the current location, a field for specifying the destination address and a button for submitting the completed form.
LCD	The touch-sensitive screen on which the map is displayed.
UpdateButton	Button used by the MobileUser and the GarageOwner to initiate the Update use case.
NavigateButton	Button used by the MobileUser to initiate the Navigate use case.
ChangeRadiusButton	Button used by the GarageOwner to initiate the ChangeRadius use case.
ChangeViewButton	Button used by the MobileUser to initiate the ChangeView use case.

2.4 Identifying Control Objects

ViewMapControl	Manages the ViewMap function. This object is created when the MobileUser selects the “OpenApplication” button. The object then creates an InputAddressForm and presents it to the MobileUser. After submitting this form, this object then collects the address information from the form and forwards it to the server. The control object then waits for the data to be submitted by the server. Once the data is received, the ViewMapControl object creates a Map and displays it on the LCD.
ViewStatsControl	Manages the ViewStat function. This object is created when the GarageOwner selects the “OpenApplication” button. The object then creates an InputAddressForm and presents it to the GarageOwner. After submitting this form, this object then collects the address information from the form and forwards it to the server. The control object then waits for the data to be submitted by the server. Once the data is received, the ViewStatsControl object creates a list and displays it on the LCD.
ChangeRadiusControl	Manages the ChangeRadius function. This object is created when the GarageOwner selects the “ChangeRadius” button. The object then creates a new List for that radius and displays it on the LCD.
ChangeViewControl	Manages the ChangeView function. This object is created when the MobileUser selects the “ChangeView” button. The object then creates a new Map and displays it on the LCD.
NavigationControl	Manages the Navigation function. This object is created when the MobileUSer selects the ‘SelectStreet” button. The object then creates a “Navigate” button which seeks confirmation from user. Now the route from current location to destination is calculated and is displayed on the LCD.
UpdateControl	Manages the Update function. The object is created when MobileUser or GarageOwner selects the “Update” button. The object sends the address information to the server. The control object then waits for the data to be submitted by the server. Once the data is received, the UpdateControl object creates a Map or List and displays it on the LCD.

3. CLASS DIAGRAM

3.1 CLASSES

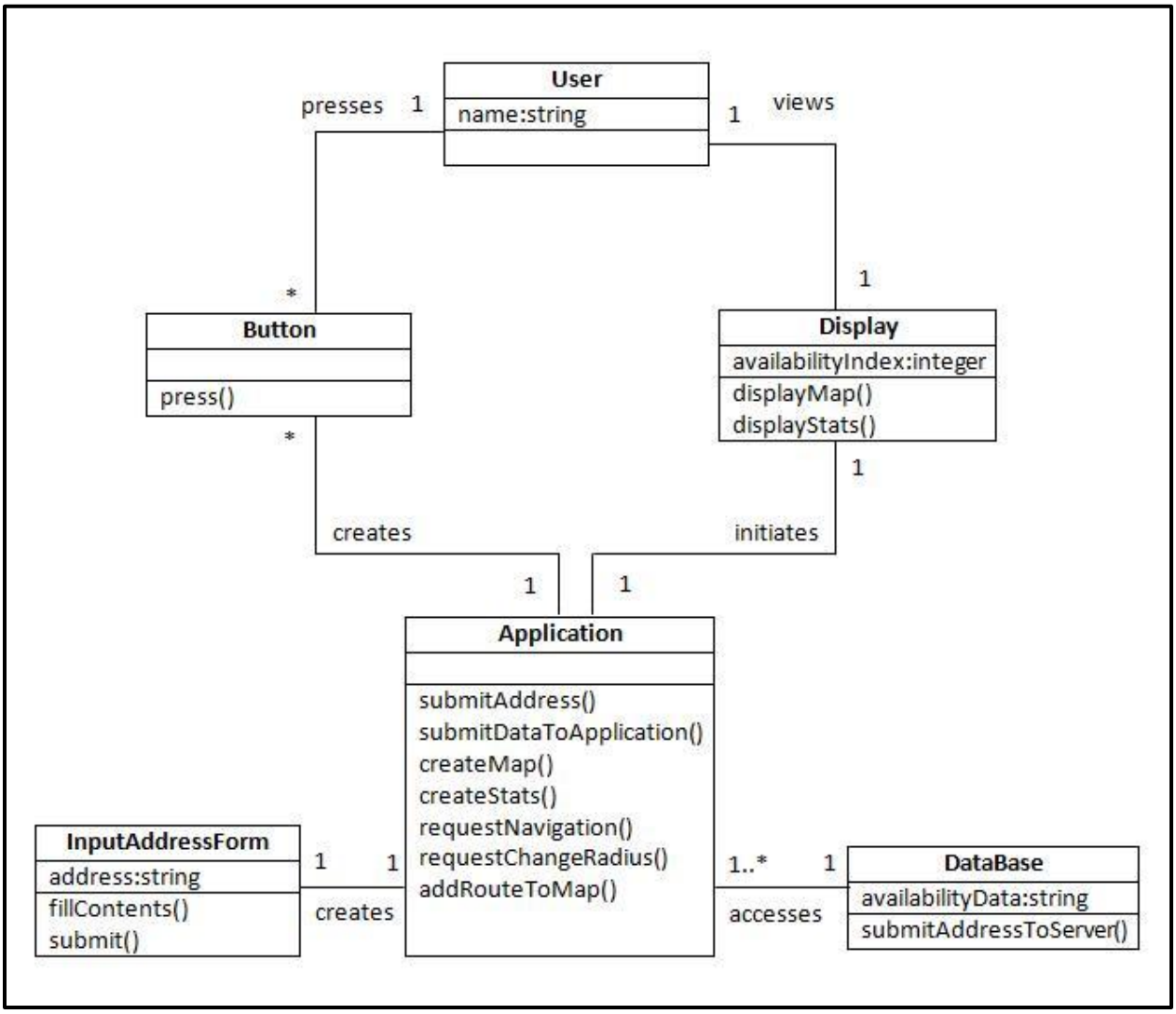


Figure 3-1 Class Diagram

3.2 INHERITANCE RELATIONSHIPS

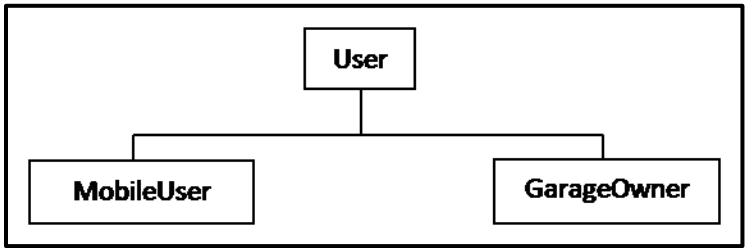


Figure 3-2 User class inheritance diagram

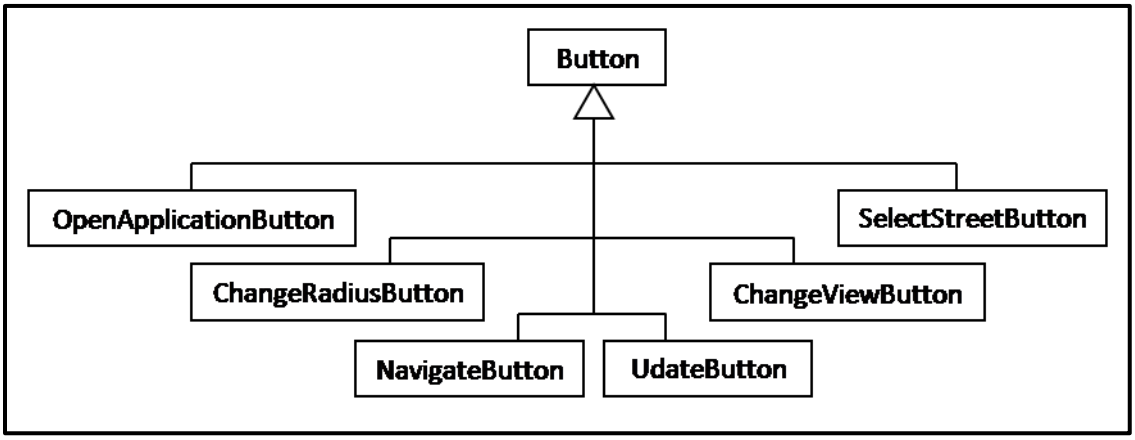


Figure 3-3 Button class inheritance diagram

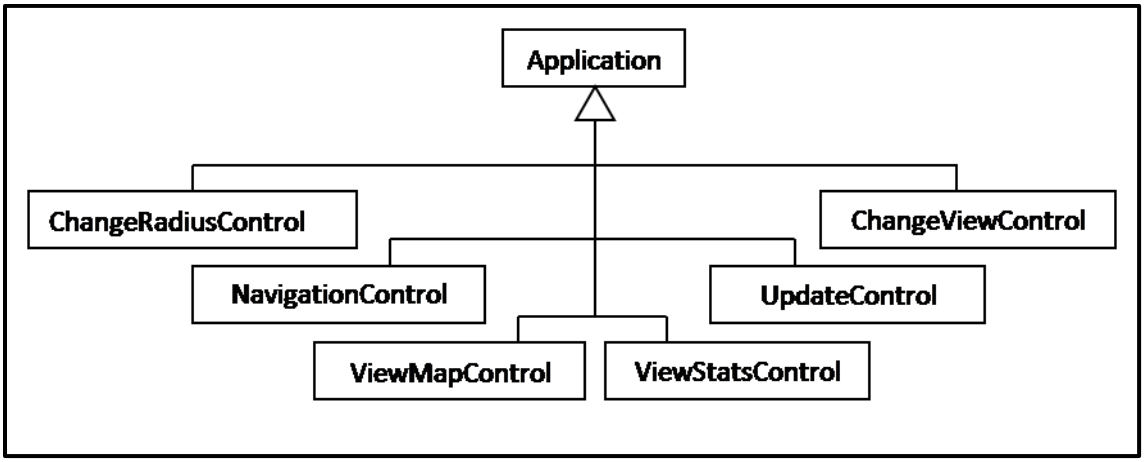
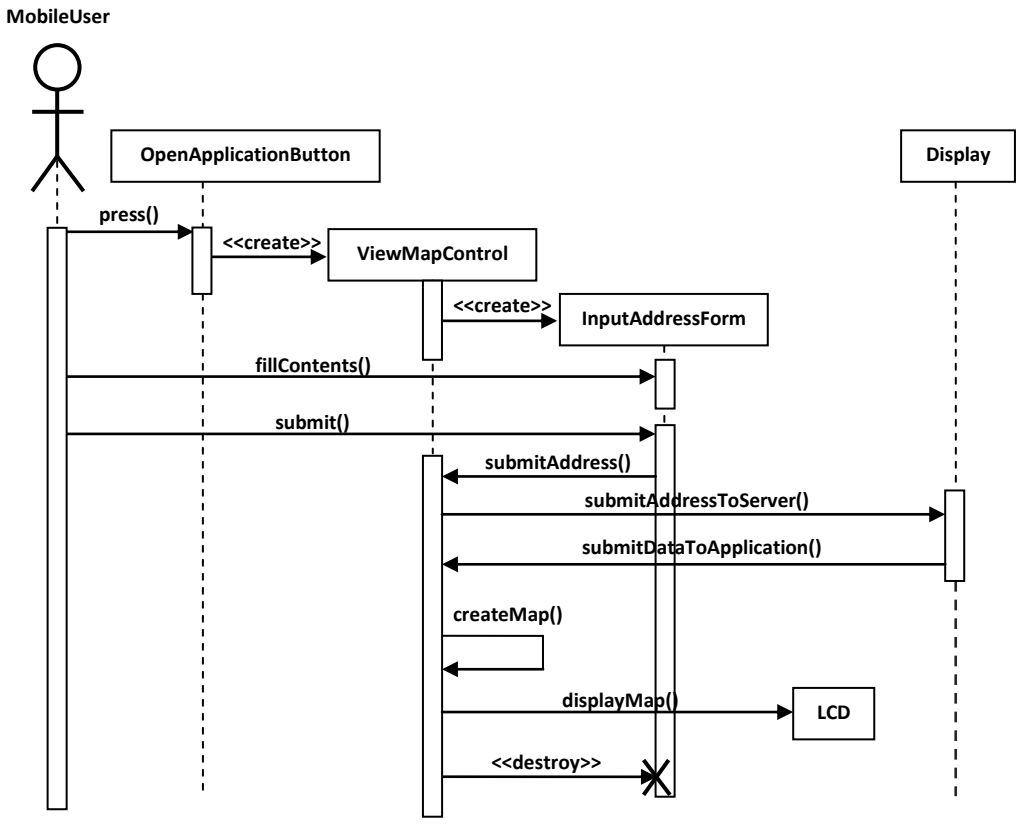


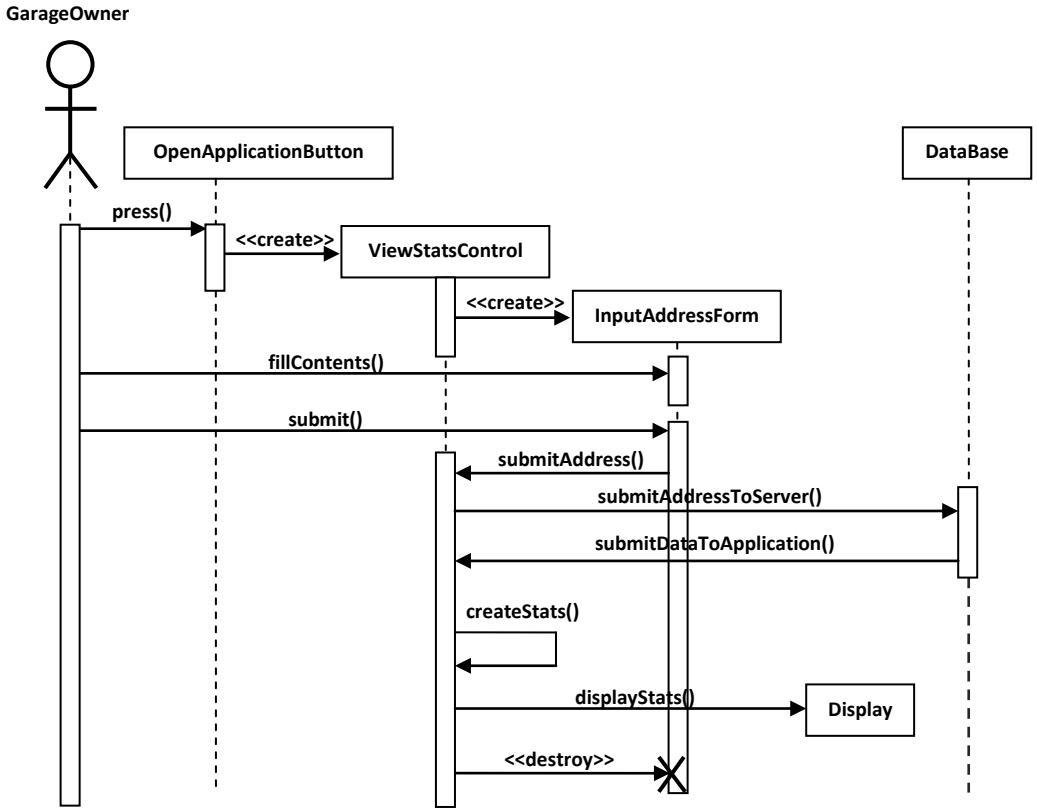
Figure 3-4 Application class inheritance diagram

4. SEQUENCE DIAGRAM

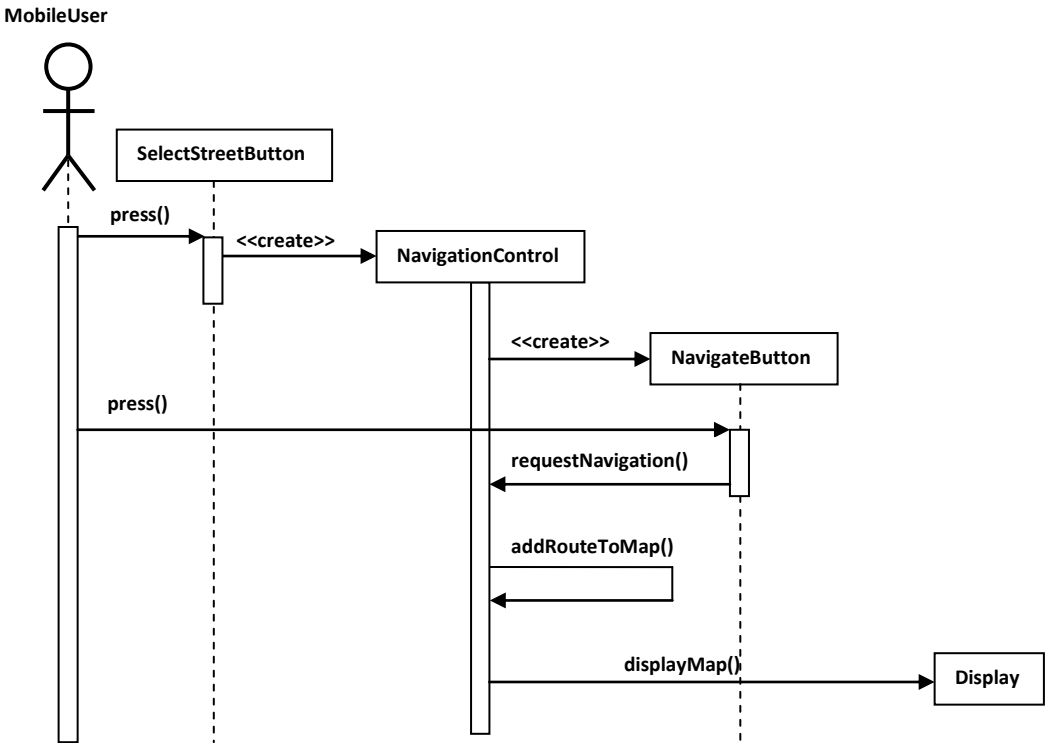
1. For ViewMap Use case



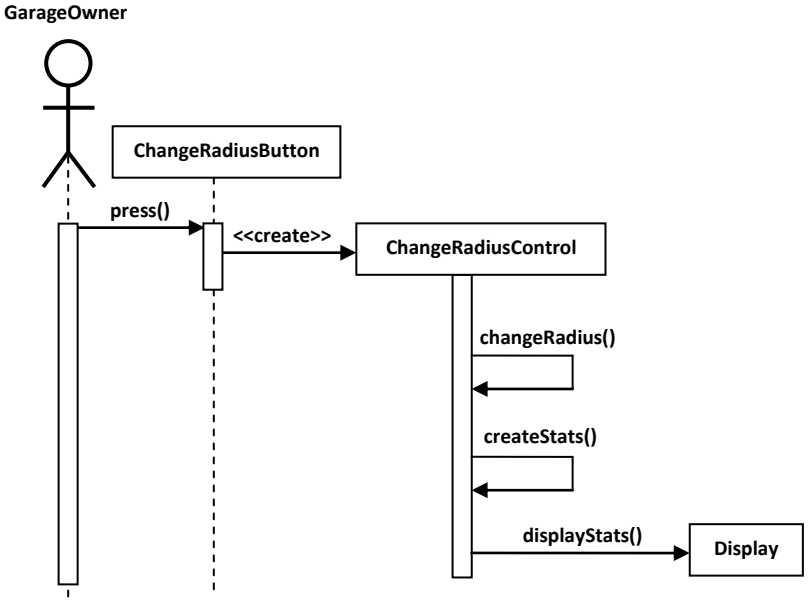
2. For ViewStats Use Case:



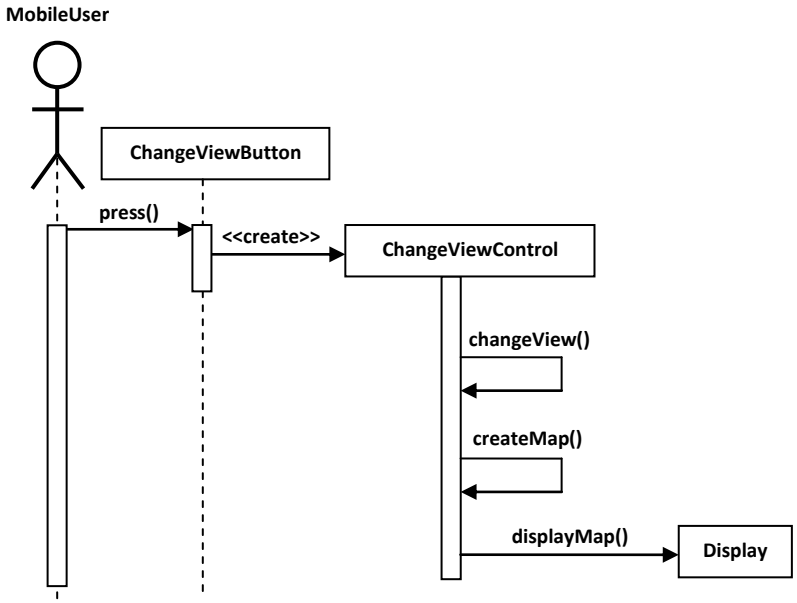
3. For Navigate Use Case:



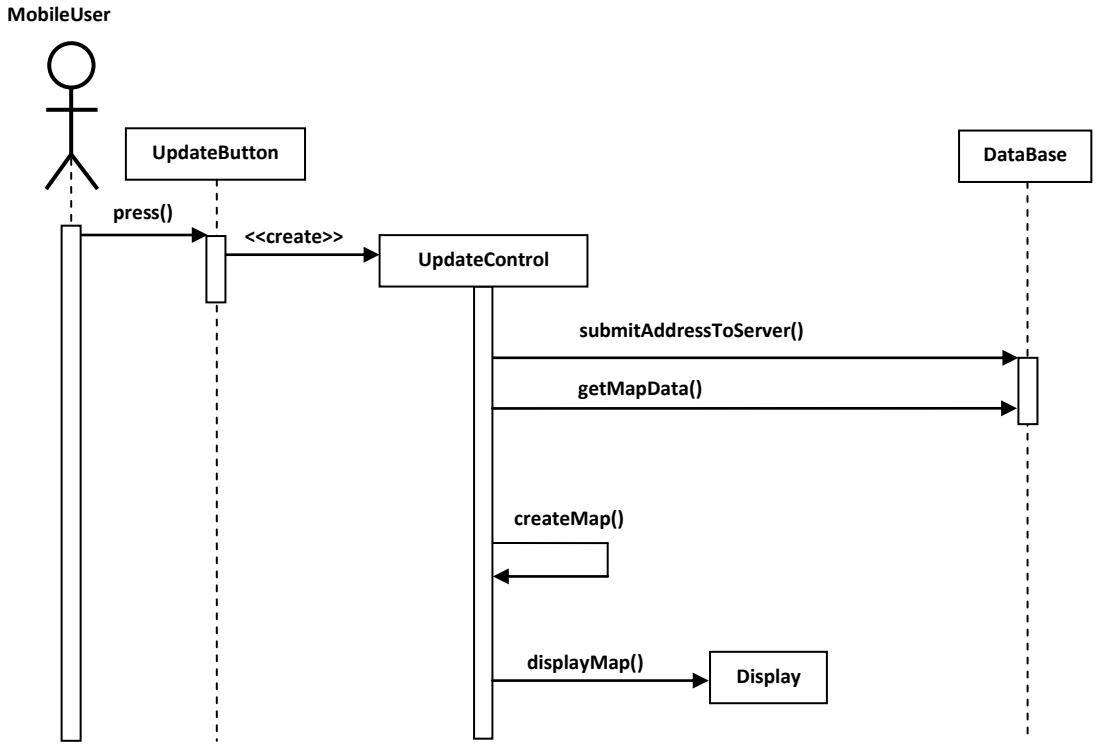
4. For ChangeRadius Use Case:



5. For ChangeView Use Case



6. For Update Use Case:



5. STATE CHART DIAGRAM