Software Requirements Specification

for

ParkEasy Mobile Application

Version 1.3 approved

Prepared by Do Anh Tu, Krithika Lakshmi

Team NoobSW

13/11/2019

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Do Anh Tu	31/10/2019	First draft for final SRS	1.0
Krithika Lakshmi	8/11/2019	Revision of First draft for final SRS	1.1
Do Anh Tu	11/11/2019	Second Revision of First draft for final SRS - Added in all components from SRS - Remove Non applicable segments from SRS - Added in formating to satisfy the document convention.	1.2
Do Anh Tu	13/11/2019	Final Submission Version - Added in Appendix B	1.3

1. Introduction

1.1 Purpose

This SRS document describes the functional and nonfunctional requirements for ParkEasy1.0. ParkEasy1.0 is a mobile application to make journey planning more convenient for users by suggesting car-parks that are close in proximity and have available vacancy to users' target destination. This document will explain the main features and design requirements of the application to relevant stakeholders and serve as an agreement before and after the development of the application.

1.2 Document Conventions

DC-1: This SRS is created with reference to the IEEE template for Software Requirement Specification Documents. Modification has been made to adapt it to the nature of a school project.

DC-2: The headings of segments in this SRS has the following specifications: Font: Arial, Size: 18 for the main heading and decrease by 2 for every subheading level under the main heading. Once the font size reach 12, the next subheading level is of size 12, Style: Bold formatted

DC-3: The content texts of this SRS has the following specifications: Font: Arial, Size: 12, Style: regular print

DC-4: The content with special attention in this document shall be **bolded**.

DC-5: References to other parts of the SRS or other documents shall be *italicized*.

DC-6: Spacing of 1.5 is added after paragraph for readability.

DC-7: As far as functional requirements concerns, nesting of number and indentation represents hierarchy of the requirements (overall - detailed - more detailed and so on).

DC-8: Bulletting or abbreviated of heading of a segment to list common ideas together.

1.3 Intended Audience and Reading Suggestions

This document is intended for developers and testers of the application who will use it as a strict guideline in their implementation and maintenance process. It will also be relevant to the course supervisors of CE2006 to better understand our project.

Readers are suggested to follow the flow of the document to understand the scope of the project, functional and non-functional requirements, interfacing involved, system features and supporting diagrams.

A Data Dictionary is provided to ensure that all readers have a clear definition of the terms that are used in this document to avoid any misunderstanding. For developers and testers - details of the use cases, class diagrams, sequence diagrams and state machine flows are attached to Appendix B that covers analysis models.

1.4 Product Scope

ParkEasy1.0 is a mobile application developed using Flutter software development kit (SDK) that can be used on both iOS and Android mobile platforms to search for car-parks that have sufficient vacant lots and are in close proximity to a user's intended destination. Users are able to search for car-parks that are upto 5km in radius from their intended destination and are able to sort their results based on their preference - by distance and vacancy. Users can then see details about the car-parks in either of two convenient views (List and Map) and are able to navigate to a car-park of their choice using an external application (Google Maps). This application also uses Google Maps Application Program Interface (API) to find suggestions for searches for destinations and uses datasets and API from data.gov.sg for information regarding car-parks in Singapore.

ParkEasy aims to ease the journey planning and driving process for its users by removing the need to worry about car-parks when looking for or driving towards their destinations. It is also a continuous effort from NoobSW team to learn good software engineering practices and designing principles.

1.5 References

The SRS for ParkEasy1.0 makes use of the following documentation:

1.5.1. Flutter SDK documentation

The documentation on Flutter SDK is founded on the website:

https://flutter.dev/

1.5.2. Government of Singapore Datasets and API

The main website for Singapore government data portal:

https://data.gov.sg/

The APIs from data.gov.sg for car-park information are founded at the webpages:

https://data.gov.sg/dataset/carpark-availability

https://data.gov.sg/dataset/ura-parking-lot

https://data.gov.sg/dataset/hdb-carpark-information

1.5.3. Google Public API

The main website for Google Developers (APIs collection):

https://developers.google.com/products/develop

The API used from Google for auto-completing for suggestions during destination input:

https://developers.google.com/places/web-service/autocomplete

1.5.4. IEEE SRS Template

A copy of the template used by this SRS can be found at:

https://ieeexplore.ieee.org/document/278253

1.5.5. Use Case Models

The use case diagram and use case descriptions for requirement analysis of this project can be obtained from the following links:

Drive: https://drive.google.com/open?id=1xfp7q0zuRKd9WQTZf32bnZAdSyRLS5kA

SVN: The documents has been compiled in Lab4_Others_Deliverables.docx in noobsw/lab4 folder of our group repository.

1.5.6. GPS Coordinate Standards

WGS-84 (used by Google)

https://www.unoosa.org/pdf/icg/2012/template/WGS_84.pdf

SVY-21 / Singapore ™ (EPSG:3414 standard)

https://epsg.io/3414

API for converting SVY21 to WGS84:
https://docs.onemap.sg/#3414-svy21-to-4326-wgs84

2. Overall Description

2.1 Product Perspective

ParkEasy is a newly developed and self-contained mobile application that will be used on smartphones that operating on Android or iOS systems. The idea for this application stems from the lack of mobile applications available to drivers in Singapore that are able to provide an estimate of vacant car-parks in the vicinity of their destination. This application will serve as a “one stop” car-park search and journey planning application - that not only

enables finding the nearest carpark, but also identifying carparks to suit the users needs in terms of estimated vacancies.

2.2 Product Functions

ParkEasy1.0 is able to perform the following functionalities:

- **Search for Destination:** user can enter their target destination (to search for car-park later) and a drop down appear in real-time, showing them matching possible destination (only in Singapore) that they are looking for.
- **Search and Show** (the possible car-parks based on destination): after finalised their destination, the user may search for a list of car-parks near to their destination. The application then:
    - Application displays car-parks within a 5km radius (by default) of the users intended destination.
- **Select a Car-park:** User can select a car-park box from List View or a car-park from Map View to view more details such as address, number of vacant lots and to be navigated to the destination.
- **Modify Displays of Results - Sort and Filter Results**:
    - Users can customize their results as per their needs by re-ordering the results with respect to distance from destination and number of lots available.
    - Users can filter results by changing radius of search (from 0km upto 5km) to suit their needs.
- **Modify Displays of Results - Alternate Views**:
    - Application offers two views fro results : List (default) and Map for users convenience.
    - Users can easily change views with a simple click of a button.
- **Navigate to Car-Park**
    - Upon selection of a car-park - the application offers to navigate to the location using an external application (Google Maps).

2.3 User Classes and Characteristics

2.3.1. Owners of Small to Medium Vehicles in Singapore

The largest segment of users for this application are expected to be local drivers of small to medium size vehicles (cars, motorcycles and vans) in Singapore that are likely to search

for car-parks in close proximity to their intended destinations. **This will be our primary users to focus on for this application development**

Some examples of this class: Taxi drivers, university students, housewives and househusbands, delivery service employees (house-movers, deliverer, posters), etc.

This user class has a diverse demographics with the common characteristics as following:

- Age: 18 years old and above
- Minimum education level: PSLE (graduated from Primary school and above)
- Frequency of use: as frequent as they drive
- Technical expertise: all members of this class should own a smartphone and are familiar using them for navigation by application such as Google Map, Waze,... It is also **assumed that the main users for this application are well-versed in interacting with common mobile application UI elements** such as buttons, dropdowns and sliders.
- Knowledges and Experiences:
    - Greatly familiar with the main areas of Singapore, especially where they work or live by. Some might have difficulties navigating in less popular area for their first time heading there.
    - Languages: English (primary) and mother tongue language
- Need: A tool to check for car-parks where they are heading for, especially where they are going for the first time.

Hence, it is important to design our application as easy to use as possible and covering a span of different users' demographics.

2.3.2. Owners of Small to Medium Vehicles in nearby states of Malaysia

There are many commuters from Malaysia that cross the Woodlands and Tuas checkpoint everyday to go to work in Singapore. Others are coming to Singapore for holiday. This group of drivers may make use of our application when they are heading to other parts of Singapore for business.

One example is factory technician who may need to travel to different factories/locations to service their machines.

This user class also has a diverse demographics, but with characteristics leaning towards to the working class:

- Age: 18 years old and above OR as permitted by Malaysian government & ICA

- Minimum education level: Not sure
- Frequency of use: daily as they commute for work
- Technical expertise: all members of this class should own a smartphone and are familiar using them for navigation by application such as Google Map, Waze, etc. They might be less technology-savvy comparing to the average Singaporean drivers.
- Knowledges and Experiences:
    - greatly familiar with the area where they work at in Singapore. They have difficulties navigating in other areas.
    - Languages: Malay and/or Chinese as primary languages
    - Having fewer understanding of Singapore road rules and legal information.
- Need: A tool to check for car-parks where they are heading for, especially where they are going for the first time.

Although they are not our main focus, the application is designed in simple UI to allow usability from all demographics.

2.3.3. Tourists Vehicle Rental Customer

This user class is very rare as they are not from Singapore nor from the surrounding countries. They have the following characteristics:

- Age: 18 years old and above OR as permitted in their international driving license
- Minimum education level: Not sure
- Frequency of use: A few times during stay in Singapore
- Technical expertise: all members of this class should own a smartphone and are familiar using them for navigation by application such as Google Map, Waze, etc.
- Knowledges and Experiences:
    - They are unfamiliar with Singapore
    - Language: English (primary)
    - Having fewer understanding of Singapore road rules and legal information.
- Need: A tool to check for car-parks and is easy to use.

Although they are not our main focus, the application is designed in simple English UI to allow intuitive usability for all tourists from different nationalities.

2.4 Operating Environment

OE-1: The ParkEasy Mobile Application will be compatible with smartphones running on **Android OS 8.0** (Oreo) and **iOS 8.0** upwards.

OE-2: ParkEasy must not have any data conflicts with other applications that make use of the Carpark API from data.gov.sg website and Google Maps public API.

OE-3: Google Maps application is required for the ParkEasy MapView and Navigate functionality.

OE-4: The target mobile device must have **internal storage of at least 120 MB**.

2.5 Design and Implementation Constraints

- **Memory**

ParkEasy retrieves data about the destination during the users input for destination in the first page from Google Maps, and suggests them in a drop down. The data about the destination is stored locally in the phone's internal storage.

The application retrieves data about the car-parks in the vicinity of the destination from data.gov.sg and this data is stored in the phone's internal storage. Thus if the phone's internal storage is full or the users wish to use secondary storage (such as SD memory cards), the application might not perform stably.

At least 120 MB of free internal storage is needed to run the app stably.

- **Cache**

The application must store the data locally each time a search is made. Once the application is closed, all data that was stored in the cache must be deleted. This allows for any real time changes there might be in the car-park details on the data.gov.sg information.

While this saves the user some memory space on their mobile phone, it costs some extra time in performance as overhead from retrieving data in each search is needed.

- **Data Standards**

When business logic databases are needed, developers must interact with data in comma separated value (.csv) format or JavaScript Object Notation (.json). Hence, any other semi-structured data format shall be converted to this format.

For geographic coordinate of the car parks, World Geodetic System 84 (WGS84) data standard shall be used with the Google Map Place API to display the location of car-parks. Developers must exercise caution with data received from the government dataset, which represent coordinates in SVY21 / Singapore ™ standard.

- **Software Architecture Design Style**

The design for ParkEasy must implement an MVC architecture where all classes are segregated into a **model** (entity classes to store data and controller classes that handle business logics like interfacing with API and processing data), **view** (user interfaces classes with states and logic on its own to capture user's input and display results to communicate with the user) and **control** (manager/logic classes that manage handling of events captured by the view classes and parse data of events to view to be handled) .

Example: Model (CarPark and CarParkManager), View(ListView and MapView,...), Control(ChangeDistanceCtrl, ChangeViewCtrl,...)

Using the MVC architecture allows for decoupling of the main UI (view and control) from the business logic (model) which makes it possible to parallely develop the system amongst the developers.

Great emphasis on developing this system with MVC has been imposed by the system designer as it will help in maintaining and scaling up the system after production.

Details of the MVC Architecture implemented in this project can be referred to in the System Architecture diagram attached in Appendix B.

- Development Toolkits and Technologies

Developers must use Android Studio IDE with Flutter SDK installed to develop the application in both iOS and Android OS.

Development and testing for Android version of this mobile application may be carried out on any computers with any variations in OS (Windows, Mac OS or Linux).

Development and testing for iOS version of the application may use the same source code as the Android version. However, testing and debugging must only be carried out on Mac OS due to limitations of the Android Studio IDE on Windows and Linux.

- Language Constraints and Programming Standards

Developers must develop the application using Flutter SDK and dart programming language.

To actualise the MVC architecture, developers must apply Object Oriented Programming Paradigm in this project.

Maintenance of the software post evaluations by the supervisors will be voluntarily amongst the team members.

- Security Standards

Developers must backup the application source code using 2 version control standards git and Subversion. This is to ensure progress checking and resolving of conflicts in code version between developers

Subversion: connect to the course main submission repository on SVN (as instructed by the course supervisor).

Only developers, software architect are allowed full access to the source code.

Tester shall be provided with relevant portions to conduct their test.

Document writers must be either a developer or the software architect to avoid mishandling of the project's source code.

2.6 User Documentation

Park Easy is straightforward and simple to use. The flow of the application is highly unidirectional from the first step of searching for user's target destination to final step of navigating to the chosen carpark by Google Map.

UD-1: **No user manual** (in paper form) is provided for the user.

UD-2: Tutorials that guide users on using the application is provided as a dialog box in the application. This **tutorial is automatically invoked when user first opens the application** (after downloading it). It can be shown to users again should they have any doubts by clicking on the information button on the application's home page.

UD-3: Alternatively, a **demo on the application usage is available on Youtube** as part of this project's submission and user may refer to its for the application's main usage.

2.7 Assumptions and Dependencies

A1: The application depends on external APIs such as data.gov.sg and Google Map for data of its core functions. If the two sources are not functioning for any reason, the application will not be able to perform its core functionalities as per normal.

A2: The application is developed using Flutter SDK such that it is only compatible for Android 8.0 version or iOS 8.0 version and upwards. It is assumed that the user will use a smart-phone that runs on either of these operating systems that are updated.

A3: It is assumed that the user has already installed Google Maps and is familiar with its usage - should the user intend to navigate to their destination through using ParkEasy.

A4: The application requires that its user is connected to a WiFi or cellular network such that it is able to access and retrieve information from data.gov.sg and google maps. It is also assumed that the user's phone location detection feature is enabled.

A5: The applications business requirements in terms of data protection and use of publicly available information are highly dependant on the current Singapore government policies as of November, 2019. Should the government change any of its policies with respect to data protection and use of available information - the system will have to be updated as necessary.

A6: The application is assumed to be used within Singapore only. Any attempt to search for car-parks elsewhere will not guarantee a positive result. Extension to other countries are only to be conducted in the future with dependency on the availability of similar data sources in those countries.

3. External Interface Requirements

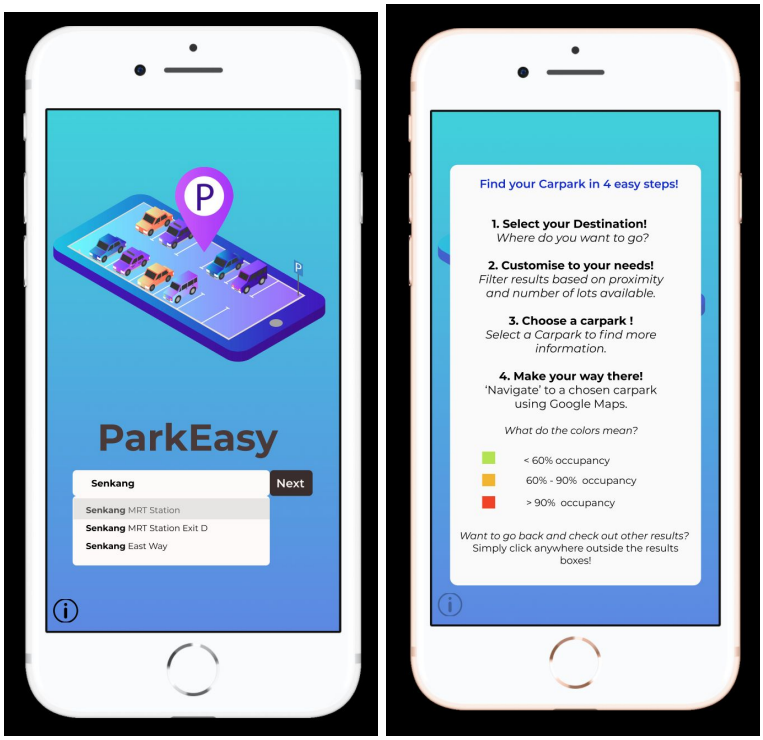
3.1 User Interfaces

A key difference between other parking applications and ParkEasy - is its highly interactive, and simple to use Interface. With dialog boxes that guide the user as to how to use the application, and simple and intuitive screen layouts - the application is easy to use.

There are four main screen layouts designed for this application that trace the user flow from entering the destination, viewing results in two alternate view formats, sorting and viewing results as per user's convenience and navigation to the selected car-park.

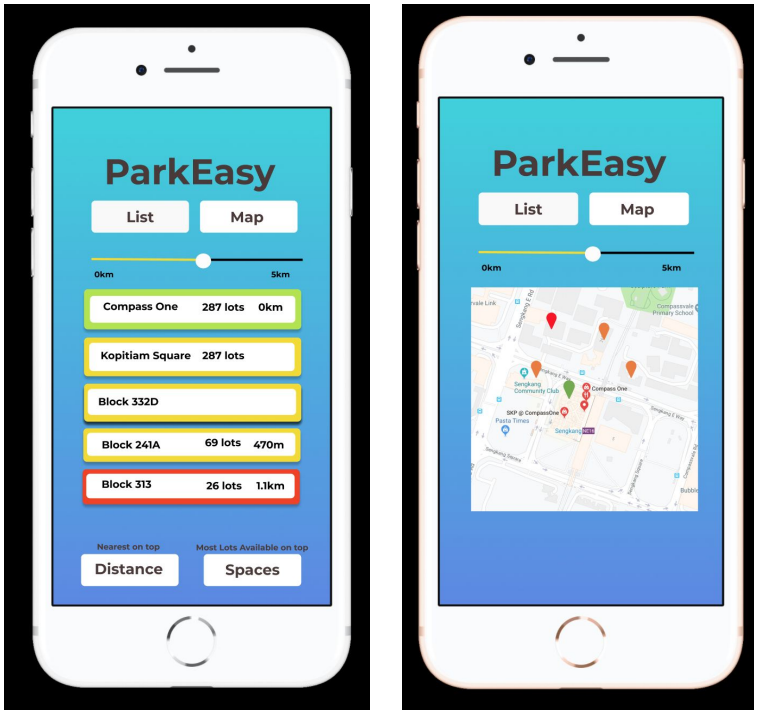
The application uses commonly seen, intuitive UI elements such as search, help, view buttons; dialog boxes and dropdown lists. To exit any open dialog boxes, the user must simply tap outside the box to return to browsing their results.

The screen layouts designed for the same are detailed below:



Screen Layout 1: User Enters their intended Destination

Uses simple and intuitive search bar, with the application offering suggestions as the user enters their destination. The user is able to select an option from the dropdown list and select “Next” to view results. The user can view a guide to using the application by clicking the commonly used “information” button on the left, bottom corner.



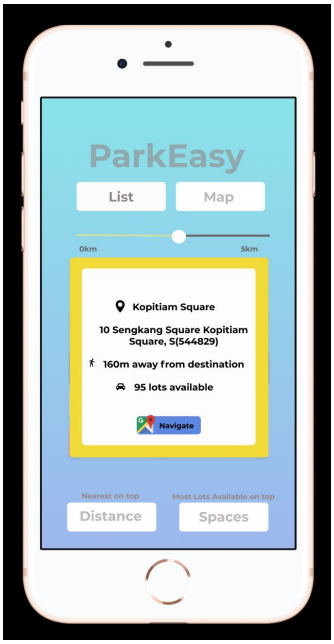
Screen Layout 2 and 3 : Alternate Views for Results and Sorting of Results

The results are displayed in two alternate views - which the user can toggle between by simply clicking on either view button at the top.

Result cards/pins are colored to indicate car-park occupancy.

The user is able to sort results in the list view using the distance/space sort buttons at the bottom of the page.

The use can filter their search radius by adjusting the distance slider at the top of the results.



Screen Layout 4 : Details of Carpark and Navigation

The user can view other details such as location, distance from destination and number of lots available when selecting a card/pin from the list/display of results.

The user can select navigate to open the location in Google Maps Application to route them to the location of their car-park.

3.2 Hardware Interfaces

To run the application, the user must use a mobile phone that operates on a minimum **Android 8.0 (Oreo) or iOS 8.0** version operating system.

The mobile phone must be connected to a WiFi network or cellular data.

The mobile phone should have enough storage space (**at least 120 MB**) to download and store data the application is processing.

3.3 Software Interfaces

For Development:

OS: Windows 10 and Mac OS X are the main computer platforms used for development

Tools: Android Studio 3.5.2, Flutter SDK 1.8.3, Android Emulator and/or Android phone with Android 8.0++ with developer mode enabled.

Data format: CSV and JSON (from APIs)

Data sharing in between classes via public variables and public classes.

For Application Functionalities and Deployment

API1: Google Maps Platform > Google Places API

- Place Autocomplete

  Request Data: Textual information with string or substring of user's destination sent from the ParkEasy application to the web service by HTTP GET Request protocol. This is to get a list of matching destination to suggest users via the Drop Down as described in Sub feature 1, Feature 4.1 of part 4 of this document

  Return Data: List of predicted matching destinations with geographical filter bounded to Singapore only is returned to the application via either JSON or XML format. This will be suggested to users and one of them will be selected as anchor point to search for car park.

- Place Details

  Request Data: An id of the destination (selected by user in Feature 4.1) sent via HTTP GET Request protocol. This is to tell Google Places API to return information about that place to us (in this case, we need the latitude and longitude of the selected destination as an anchor point for searching car-parks

  Return Data: JSON file containing information about the selected destination. This will be used to focus the search area of car-parks.

API2: Carpark Availability API from Data.gov.sg

Request Data: car-park availability information of all car parks in Singapore. This is a HTTP GET Request and is combined with API3: datasets of carpark in Singapore for display of car-park search results in Feature 4.1, 4.3 and 4.5 of the application described in part 4 of this SRS.

Return Data: car-park availability information, updated in real-time. Data format is JSON.

API3: Dataset for Details of Car-parks under Housing Development Board (HDB) in Singapore

Request Data: fetch and download the csv file dataset from the Data.gov.sg repository when the user start the application

Return Data: Full car-parks' details in .csv format. This is used together with API 2 to display all information about relevant car parks in Feature 4.1, 4.3 and 4.5 of the application as described in part 4 of this SRS.

API4: SVY21 To WGS84 Converter API for Car Park coordinates by OneMap

Request Data: POST HTTP Request (supplying x and y coordinates in URL of request) with the SVY21 format coordinate of the car-park is sent to the REST API service. This format for coordinate is only used by Singapore and we need to convert to something that Google Places can understand.

Return Data: WSG84 coordinate of the car park. This conversion is incorporated in processing the .csv file of HDB carpark dataset (API 3) to the final data source to be used by the application as this converted data source is compatible with Google Places API.

3.4 Communications Interfaces

ParkEasy requires a steady internet connection to load data from data.gov.sg Car-park Availability API and Google Maps API to get the required information on car-parks and their availability, and the details about the destination.

HTTP GET protocol is the main mode of communication between our application and the relevant, aforementioned APIs

4. System Features

The system features are based on use cases that are grouped by their functionalities.

For detailed use case descriptions of each sub-feature, please refer to the Use Case Descriptions and Use Case Diagram documents and how to obtain them as stated in 1.5. References

4.1. Enter a destination and View search results of car-parks in List View (default display)

4.1.1. Description and Priority

This segment describes the feature of the application that allows the user to enter their intended destination to view car-parks in proximity of the destination. This feature is of high priority, since the application would not be able to perform its core functions without getting a user input initially. There are two major actions involved in the process to enter destination and view results.

- Sub-Feature 1 :
    - User should be able to enter their intended end destination using the application. (priority number : 1)
    - Corresponding Use Cases: *Enter Destination* (ID: 1.12)
- Sub-Feature 2 :
    - The application shows the list of results (using Government API) based on the previously entered destination and user can select the destination. (priority number: 1)

- Corresponding Use Cases: *Enter Destination* (ID: 1.12), Search car-parks (ID: 1.15)

4.1.2. Stimulus/Response Sequences

Precondition : The application is open and on the first page.

4.1.2.1.Sub - Feature 1: Enter Intended Destination

Stimulus : User is prompted to enter their intended destination.

Response : The user starts typing their intended destination.

Response : The application retrieves data from the Google API to display a list of suggestions based on users input.

Stimulus : The application displays a dropdown list of suggestions based on user input.

Response : User selects a destination from the dropdown list.

Stimulus : Next Button is “highlighted”

Response: User selects “Next” button.

4.1.2.2. Sub - Feature 2: Display results

Stimulus : Next button is selected.

Response : The application opens a new page with the default list view of results based using data retrieved from the Government API.

4.1.3. Functional Requirements

1. On the first page, the app must ask the user for the intended destination
2. The destination must be typed out by the user.

    2.1. As the user types in the destination, the app must show a dropdown list of locations matching what the user has written

    2.1.1. The list of locations must be received from the Google Maps API

        2.1.1.1. The locations must only be from Singapore.

    2.2. After the user has selected a location from the dropdown, a “Next” button must be highlighted.

    2.3.When the user presses the next button, the app must switch to a new page.

3. The next page must show the car-parks(based on Government API) near the destination entered in the previous page

    3.1.There must be 2 buttons at the top of the page

        3.1.1. The first button must say “List”

            3.1.1.1.This button must be selected by default

            3.1.1.2.Pressing this button must show the car-parks in list format

        3.1.2.The second button must say “Map”

            Pressing this button must show the car-parks in map format

    3.2. By default, the list view must be shown.

        3.2.1. The car-parks must be ranked by distance from the entered destination(lowest to highest) by default.

    3.3. In the list, each car-park must be shown as a seperate card

        3.3.1. Each card must display the car-park’s number, name, number of spaces available, and distance from destination

        3.3.2. The border of each card must denote the vacancy of the car-park by colour coding

            3.3.2.1.The border’s colour must be green to indicate that the car-park has occupancy lower than 60%

            3.3.2.2.The border’s colour must be orange to indicate that the car-park has occupancy between 60% and 90%

            3.3.2.3.The border’s colour must be red to indicate that the car-park has occupancy greater than 90%

4.2 Dynamically Modifying List View Based On User’s Preference

4.2.1. Description and Priority

This segment describes the main features of the system feature of modifying the list view dynamically at runtime based on user’s selected preferences. There are three actions the users can carry out to modify the List View to their preference:

- Sub-feature 1:

- Change the range to search for car-park. (priority number 2)
- Corresponding Uses Case: *Change Search Distance* (1.19)

- Sub-feature 2:
  - Change the sort order of car-park for displaying on list view. (priority number 2)
  - Corresponding Uses Case: *Change Sort* (1.20)
- Sub-feature 3:
  - Change the display view of car-parks to Map View. (priority number 1)
  - Corresponding Uses Case: *Change Display Type* (1.17)

4.2.2. Stimulus/Response Sequences

4.2.2.1. Subfeature 1: Change the range to search for car-park.

Preconditions: The system is in List View

Stimulus: User slides the slider pointer to desired search range.

Response: System filters out of range car-parks and display those that are in searching range on the list view.

4.2.2.2. Subfeature 2: Change the sort order of car-park for displaying on list view.

Preconditions: The system is in List View

Stimulus 1: User clicks on Distance button

Response: System orders list view of car-parks by closest distance to the selected destination first.

Stimulus 2: User clicks on Spaces button

Response: System orders list view of car-parks by the most vacant first.

4.2.2.3. Subfeature 3: Change the display view of car-parks to Map View.

Preconditions: The system is in List View

Stimulus: User clicks on Map button

Response: System switch active car-park display UI from List View to Map View

4.2.3. Functional Requirements

1. In the current active List View, there must be a slider on top of the list view page to allow user changing searching range (sub-feature 1)

    1.1. The slider must indicates the current range of distance for searching car-parks

    1.2. When the user slides the slider, only car-parks within the distance on the slider from the destination must be shown in the list.

    1.3. The change in car-park listed should happens instantaneously.

        1.3.1. The updated list of car-park is shown the moment the user stop their fingers on the preferred final position of the slider pointer.

    1.4. The minimum slider value must be 0km and the maximum must be 5km.

2. There must be 2 buttons at the bottom of the current List View page to allow sorting of car-park list in List View (sub-feature 2)

    2.1. The first button must have “Distance” as it text.

        2.1.1. When pressed, the button must sort the car-parks based on distance from destination.

            2.1.1.1. car-parks with shorter distance to user’s destination are displayed first.

            2.1.1.2 The Distance button must be selected by default

    2.2. The second button must have “Space” as it text.

        2.2.1. The second button must sort the car-parks based on number of spaces available.

            2.2.1.1. car-parks with higher vacancies must be displayed first.

            2.2.1.2. car-parks with lower vacancies are displayed later.

3. There must be 2 buttons at the top of the List View page to allow for switching between List and Map View (sub-feature 3)

    3.1.The first button must display “List” on itself

        3.1.1. This button must be selected by default

        3.1.2. Pressing this button must show the car-parks in list format

    3.2.The second button must display “Map” on itself

        3.2.1. Pressing this button must show the car-parks in map format

4.3 Display Search Results in Map View

4.3.1. Description and Priority

This segment describes the feature of the application that displays the Map View presentation of the search results for car-parks based on destination selected (refer to sub-feature 1 and 2 of part *4.1. Enter a destination and View search results of car-parks in List View (default display)* for more info)

- The application shows the map view of results of car-parks (using Government API) based on the previously entered destination (priority number: 1)
- Corresponding Use Cases: *Display car-park* (ID: 1.18)

4.3.2. Stimulus/Response Sequences

Preconditions: The Map View button must be clicked by user.

Stimulus : Map View button active state (pressed) is captured and notified to Map View

Response : The application opens a new page with the map view of results based using data retrieved from the Government API.

4.3.3. Functional Requirements

1. The map view must show a map with the entered destination at the center

    1.1. All car-parks within a 5km radius of the destination must be shown

        1.1.1. The car-parks must be shown as pins on the map

            1.1.1.1. The pins must display the number of empty spaces in the car-park

            1.1.1.2. The colour of the pin must denote the emptiness of the car-park.

                1.1.1.2.1. The pin’s colour must be green to indicate that the car-park has occupancy lower than 60%

                1.1.1.2.2. The pin’s colour must be orange to indicate that the car-park has occupancy between 60% and 90%

                1.1.1.2.3. The pin’s colour must be red to indicate that the car-park has occupancy greater than 90%

4.4. Dynamically Modifying Map View Based On User's Preference

4.4.1. Description and Priority

This segment describes the main features of the system feature of modifying the map view dynamically at runtime based on user's selected preferences. There are two actions the users can carry out to modify the Map View to their preference:

- Sub-feature 1:
    - Change the range to search for car-park. (priority number 2)
    - Corresponding Use Cases: Change Search Distance (1.19)
- Sub-feature 2:
    - Change the display view of car-parks to List View. (priority number 1)
    - Corresponding Use Cases: Change Display Type (1.17)

4.4.2. Stimulus/Response Sequences

4.4.2.1. Subfeature 1: Change the range to search for car-park

Precondition: The application must be in Map View

Stimulus: User slides the slider pointer to desired search range.

Response: System filters out of range car-parks and display those that are in searching range on the map view.

4.4.2.2. Subfeature 2: Change the display view of car-parks to List View.

Precondition: The application must be in Map View

Stimulus: User clicks on List button

Response: System switch active car-park display UI from Map View to List View

4.4.3. Functional Requirements

1. In the current Map View, there must be a slider on top of the map view page to allow user changing searching range (sub-feature 1)

    1.1. The slider must indicates the current range of distance for searching car-parks

1.2. When the user slides the slider, only car-parks within the distance on the slider from the destination must be shown on the map.

1.2.1. Number of pins on the map must correspond to the number of car-parks found.

1.3. The change in number of car-park pins on the map should instantaneously.

1.3.1. The updated map view of car-park is shown the moment the user stop their fingers on the preferred final position of the slider pointer.

1.4. The minimum slider value must be 0km and the maximum must be 5km.

2. There must be 2 buttons at the top of the current map view page to allow for switching between List and Map View (sub-feature 2)

2.1.The first button must display “List” on itself

2.1.1. This button must be selected by default.

2.1.2. Pressing this button must show the car-parks in list format.

2.2.The second button must display “Map” on itself

2.2.1. Pressing this button must show the car-parks in map format.

4.5. Select car-park

4.5.1. Description and Priority

This segment describes the feature of the application that allow user to select a car-park from a list of car-parks (in List View) or from a cluster of pins (in Map View). When selected, a dialog box containing full information about the car-park will be shown to the user.

- Sub-feature 1: The user can select and view the details of a selected car-park from list. (priority number 1)
- Sub-feature 2: The user can select and view the details of a selected car-park from a pin on the map view. (priority number 1)
- Corresponding Use Cases: *Select car-park* (ID: 1.21)

4.5.2. Stimulus and Response

4.5.2.1. Sub-feature 1: Select and View the details of a selected Car-Park from list

Stimulus : User selects a single result from the list view.

Response : The application displays a dialog box that details all the information about the car-park and present the choice to navigate to the car-park selected

4.5.2.1. Sub-feature 2: Select and View the details of a selected Car-Park from Map View

Stimulus : User selects a single result from a pin of the map view.

Response : The application displays a dialog box that details all the information about the car-park and present the choice to navigate to the car-park selected

Stimulus : The user clicks outside the dialog box.

Response : The application exits the display box and returns to the list view of results.

4.5.3. Functional Requirement

1. The dialog box must have a button at the bottom which says “Open in Google Maps”

   1.1. When pressed, the button must open the Google Maps app with the selected car-park’s address as the location

   1.2. If the user decides to go back to the app from Google maps, the app must show the search results in the view before the user opened the dialog box.

4.6. Navigate to a selected car-park

4.6.1. Description and Priority

This use case assumes that the user has selected a car-park to view details and describes the feature of the application that allows the user to navigate to the location using an external application. This feature is of low priority.

- User can navigate to the car-park using an external application - Google Maps (priority number : 3)
- Corresponding Uses Case: Navigate (1.22)

4.6.2. Stimulus and Response

Precondition : The dialog box that displays the details of a selected car-park (from the results views) is open.

Assumption : The user has already installed Google Maps on the local system.

Stimulus : The user selects “Navigate” on the dialog box.

Response : The application exits to open the location of the selected car-park in Google Maps.

Stimulus: User decides to go back to the application from google maps.

Response : The application shows the search results in the view selected prior to user opening the dialog box.

4.6.3. Functional Requirement

1. The dialog box must have a button at the bottom which says “Open in Google Maps”

   1.1. When pressed, the button must open the Google Maps app with the selected car-park’s address as the location

   1.2. If the user decides to go back to the app from Google maps, the app must show the search results in the view before the user opened the dialog box.

4.7 Help and Tutorial for User

4.7.1 Description and Priority

This segment describes the feature that allows users to use a “guide” as a tutorial on how to use the application. This feature is of low priority, as it may not be used as frequently or critically as other features.

- User can view tutorial dialog box. (priority number : 2)
- Corresponding Uses Cases: Get Help (1.24)

4.7.2. Stimulus/Response Sequences

Precondition : The application is open.

Stimulus : User clicks “information button” on the bottom left corner.

Response : The system opens a dialog box that explains the four steps involved in using the application.

Stimulus : User clicks outside dialog box.

Response : The application exits the tutorial dialog box and returns to standard view when application is first opened.

4.7.3. Functional Requirements

1. The first page must have an “Info” button at the bottom left

1.1.Pressing the info button must open up a dialog box

1.2.The dialog box must explain the four steps of using the app

1.2.1. The dialog box must show the first step as selecting a destination

1.2.2. The dialog box must show the second step as filtering list of car-parks

1.2.3.The dialog box must show the third step as selecting a carpark

1.2.4.The dialog box must show the last step as opening the car-park in google maps.

1.3.The dialog box must tell the color coding used by the app for vacancy levels.

1.EX.1. The user can exit the dialog box by clicking outside the box.

1.EX.1.1. The application returns to the standard view of the first page when application is opened.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

During regular performance:

- The application must be available to its users at least 99.9% of the time between the hours of 7AM and midnight
- The application must respond to the user requests in less than one second.
- The application must acknowledge all user queries within 30 seconds.
- The application must enable at least 80% of first time users to enter a search query within two minutes of starting to use the system.
- Querying of the government API for parking spaces must take less than 1 second.
- The app must be able to support at least 50 queries per minute.

In the case of failure/reboot:

- 99.9% of all app system failures must be repairable in under one hour.
- The app must run without failure for at least 24 hours after being restarted, under normal conditions of use.
- After a system reboot, the app must restore full functionality within 1 minute.

5.2 Security Requirements

- The app shall not store/share data to third parties regarding the location of the user and car-parks used.
- The app shall not share user's history of the app usage to third party companies.

5.3 Software Quality Attributes

ParkEasy promises to achieve quality especially with respect to -

- Reliability :

  ParkEasy will provide accurate and correct results for user queries. The white and blackbox testing for the application has proven that there are few known bugs and a low error rate.

- Reusability and Extensibility :

  ParkEasy is implemented in a manner that allows for extension of the application in the future to include more functionalities - possibly paying for car-parking and real time route planning during a journey. In using MVC architecture, the implementation separates the UI from the core data model. ParkEasy currently makes use of two main user interface views - List and Map views.

  By separating the data from UI - it makes the data model and business logic reusable across both the interfaces. Should the project be chosen for further extension by changing/adding new UI features - the MVC architecture makes it easy for developers to redesign the UI without having to significantly affect other functionalities of the application.

- Availability :

  The application guarantees to be available to its users at least 99.9% of the time between the hours of 7AM and midnight. It will function fully provided there is a steady internet connection and the users "use location" settings are enabled.

- Maintainability :

  Since the implementation uses an MVC architecture to decouple the UI logic from the data model and core business functions - it makes it possible to easily maintain and fix any bugs as they will be isolated to their specific components. The entire design, implementation and testing of the application has been documented in detail including rationale for design choices and detailed analysis models for programmers and testers reference.

6. Other Requirements

Not applicable in this submission

Appendix A: Glossary

This is a glossary for terminology used in this documentation.

Term	**Definition**
Car-park	An area or a building where cars or other vehicles may be left temporarily
Destination	The place to which the user is going
App / Application	Software program that runs on a computer and provides some service
API (Application Programming Interface)	A software intermediary that allows two applications to talk to each other, which here is the Carpark availability government API and Google Maps API
Google Maps	An application by google to show geographical sites, regions and relevant information about it
Page	The smaller part of an app representing the 2 different features which are the opening page for location selection and list of parkings
Government API	An application interface developed by the Government for developers to use with their platform

Map	A diagrammatic representation of an area of land or sea showing physical features, cities, roads, etc brought to us by Google via their application Google Maps
Slider	A graphical control element with which a user may set the values of the range of finding car-parks nearby by moving an indicator horizontally upto 5 kilometers for range.
Card	A container within a page representing the location and details about the parking
Dropdown list	A graphical user interface element, that allows the user to choose one value of the location of the car-park from a list
Border	The outline of the pins and cards, for which different colours represent different occupancy levels, with green meaning the occupancy level is lower than 60%, orange means that the occupancy level ranges between 60% to 90% and red meaning that the occupancy is more than 90% of the car-park selected.
Dialog box	A type of window that is used to enable communication and interaction between a user and a software program.
Pins	Inverted-drop-shaped icon that marks locations in Google Maps
MVC	A software architecture style that supports interaction based applications - by decoupling UI logic from the core business logic to allow for reusability and extensibility. Involved classes being categorised as Model (storage classes),

	View (UI logic classes) and Controller (request handling classes).

Appendix B: Analysis Models

The following documents are available in the order stated below:

1. Class Diagram
2. Sequence Diagram (by Features as described in Part 4)
    a. Get Help / Giving Tutorial to users
    b. Search Destination
    c. Search Car Park
    d. Modifying ListView
    e. Modifying MapView
    f. Select Car Park (List View)
    g. Select Car Park (Map View)
3. State Diagram
    a. Search Destination and CarPark
    b. List View
    c. Map View

SelectCarParkCtrl
+handleSelectedCarPark(CarPark: seleected)

Control

<<create>>

ChangeViewCtrl
+handleChangeView(int viewType): int

CarPark
+number: String
+address: String
+lotsAvailable: int
+totalLots: int
+fractionTaken: double
+distance: double
+latitude: double
+longitude: double

* Data

Biz Logic 1 Model

ChangeDistanceCtrl
+handleChangeDistance(double range)

Control

MapView
+carParkList: List<CarPark>
+allMarkers: List<Markers>
+selectedarPark: CarPark
+colour: double
+selectCarPark(CarPark c)

<<Interface>> GovtDataAPI
getCarParkAvail(): String response

<<calls>>

CarParkManager
+allCarParkList:List<CarPark>
+neededCarParkList: List<CarPark>
+selectedCarPark: CarPark
+sortType: int
+observers: List<ParkEasyObserver>
+attachCount:int
+getAllCarParkList()
+getNeededCarParkList(double range)
+sortCarParkList(int sortType)
+selectCarPark(CarPark: cp)
+getDistance(double lat1, lat2, lon1, lon2): double

Model Control

ChangeSortCtrl
+handleChangeSort(int sortType)

CarParkCard
+carPark: CarPark
+borderColor: Color

<<Interface>> GovtDataSet
GetCarParkCoordinate()

<<calls>>

ListView
+carParkList: List<CarPark>
+sortType: int
+selectedCarPark: CarPark
+changeSort(int sortType)
+selectCarPark(CarPark c)

DisplayCarParksView
+changeView(int type)
+changeDistance(double d)

<<Interface>> ParkEasySubject
+attach (o: ParkEasyObserver)
+detach (o: ParkEasyObserver)
+notify()
+getData(): List

Model

for each o in registered observers o.update()

Observer

<<Interface>> ParkEasyObserver
+update()

SpaceSortButton
+selected: bool

MapViewButton
+clicked

ListViewButton
+clicked

DistanceSlider
+range: double
+rangeLabel: String

DistanceSortButton
+selected: bool

<<Interface>> GoogleAPI
+getLatLong(Destination d): Destination
+getMatchingLocations(String i): List<Destination>

<<calls>>

DestinationManager
+destinationList: List<Destination>
+selectedDestination: Destination
+getDestinationList(String text): List
+selectDestination(Destination selected)

Model Control

SearchDestCtrl
+handleEnterDestination(String text)
+handleSelectDestination(Destination s)

SelectedDialogBoxView
+borderColor: Color
+carPark: CarPark
+navigate(double lat, double lon)

<<creates>>

NavigateCtrl
+handleNavigate(double: lat, double lon)

Biz Logic 1

Data *

Destination
+lattitude: double
+longtitude: double
+name: String
+placeID: String

creates

SearchDestView
+selectedDest: Destination
+listOfDest: List<Destination>
+enterDestination(String text)
+selectDestination(Destination d)
+showInfoDialogBox()

Startup
+main()

InfoDialogBoxView

<<creates>>

ShowInfoButton
+selected

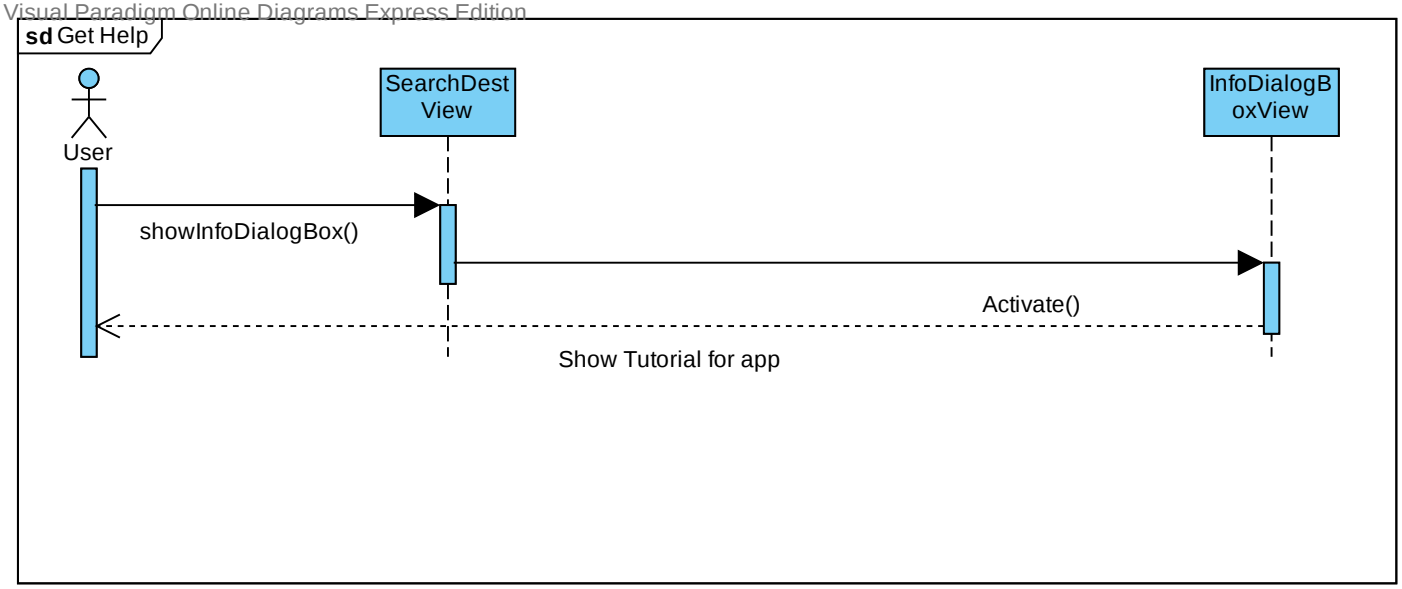
NextButton
+active

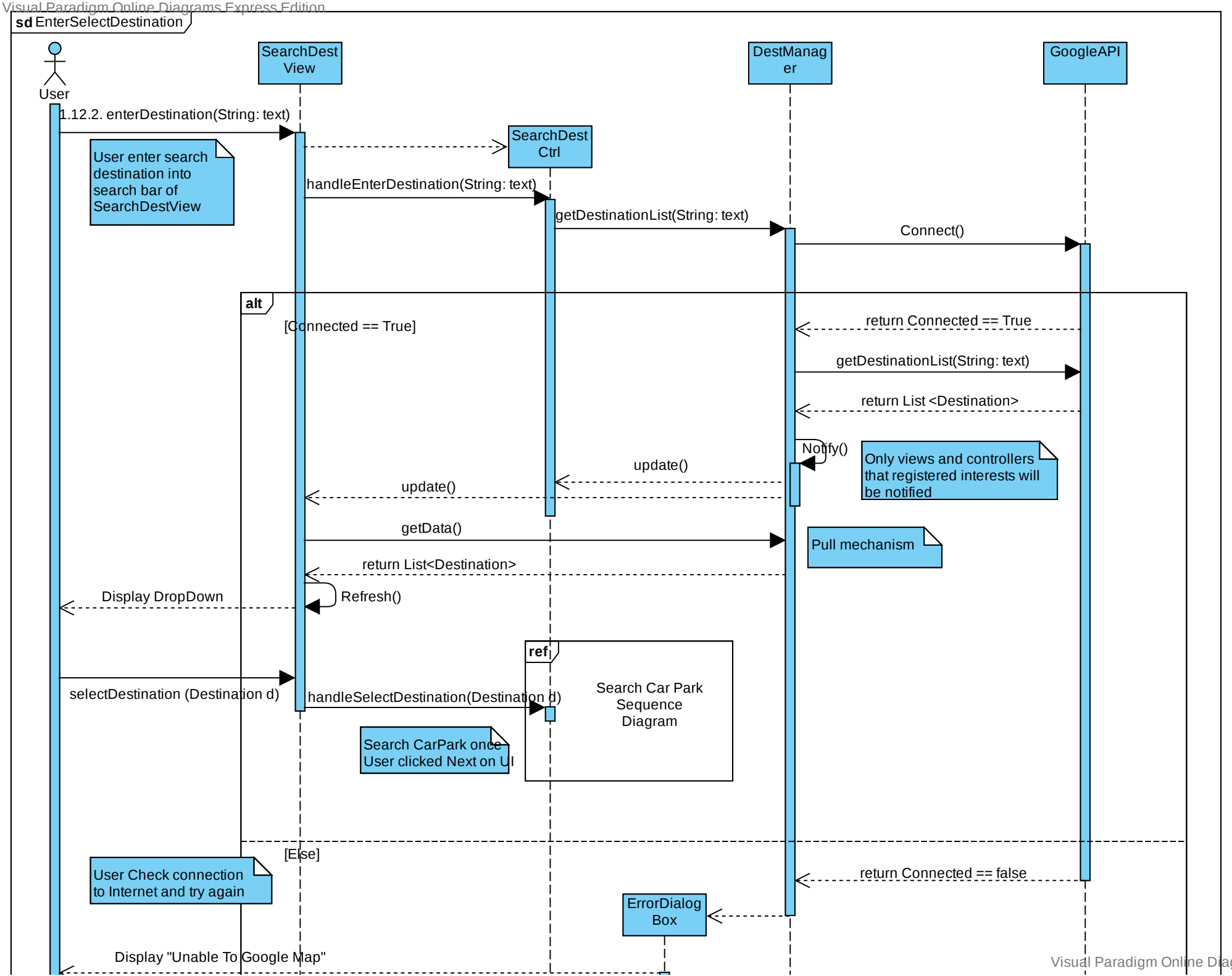
DropDown
+listDestination: List<Destination>
+selectedDestination: Destination

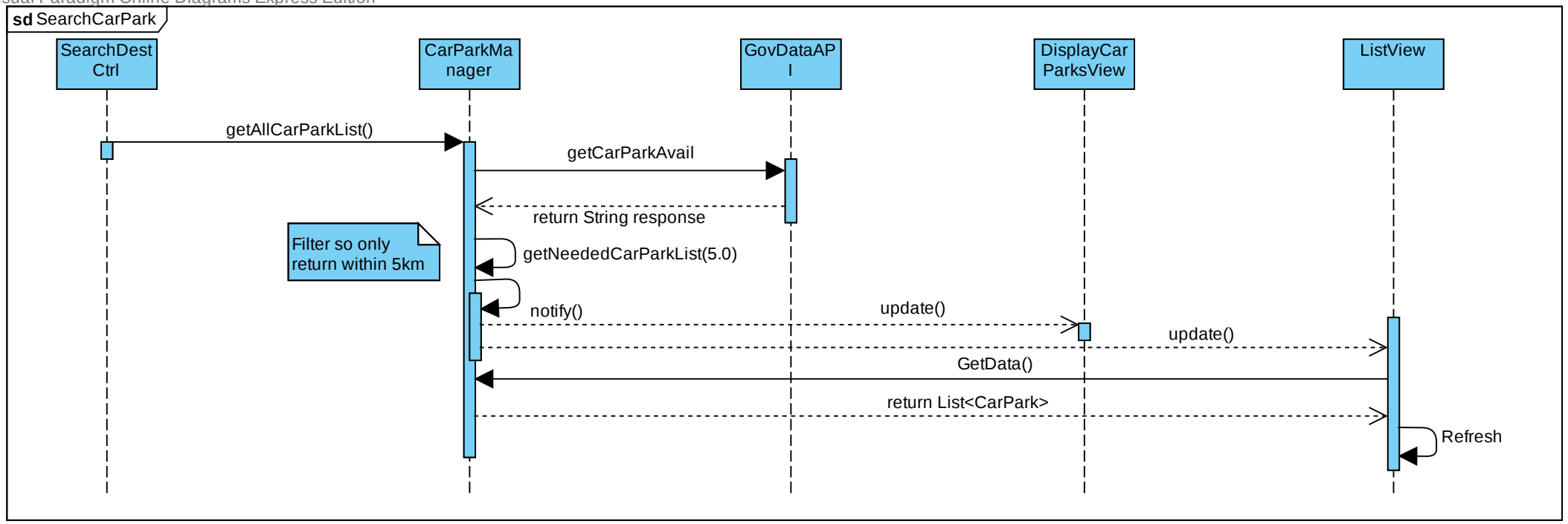
SearchBox
+text:String

Note on association:
By Default, association arrow denotes 1 to 1 relationship, unless other specified in the diagram

<<calls>>



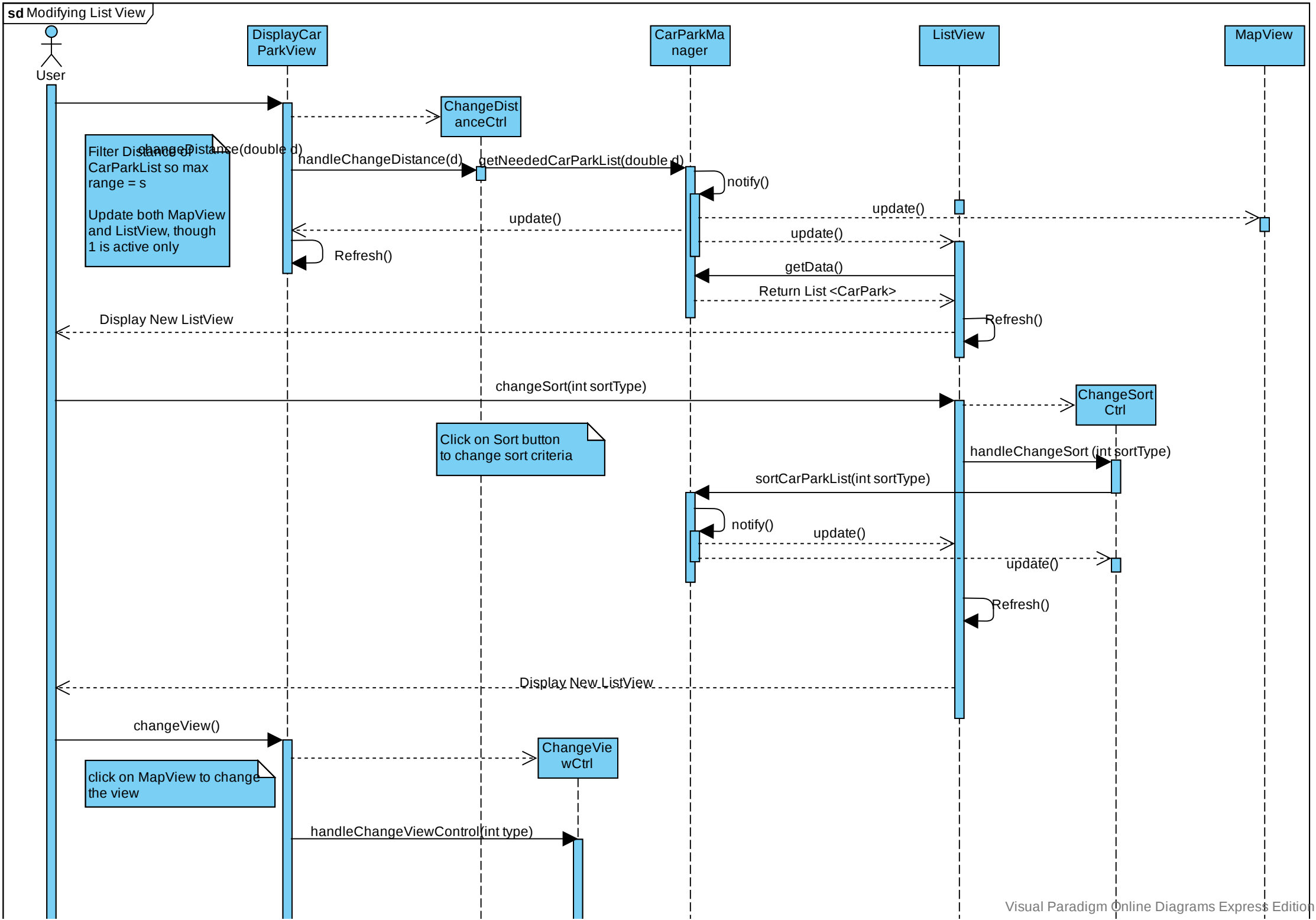


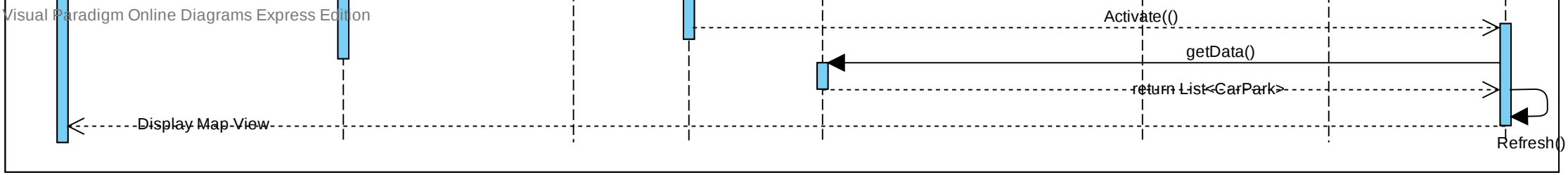


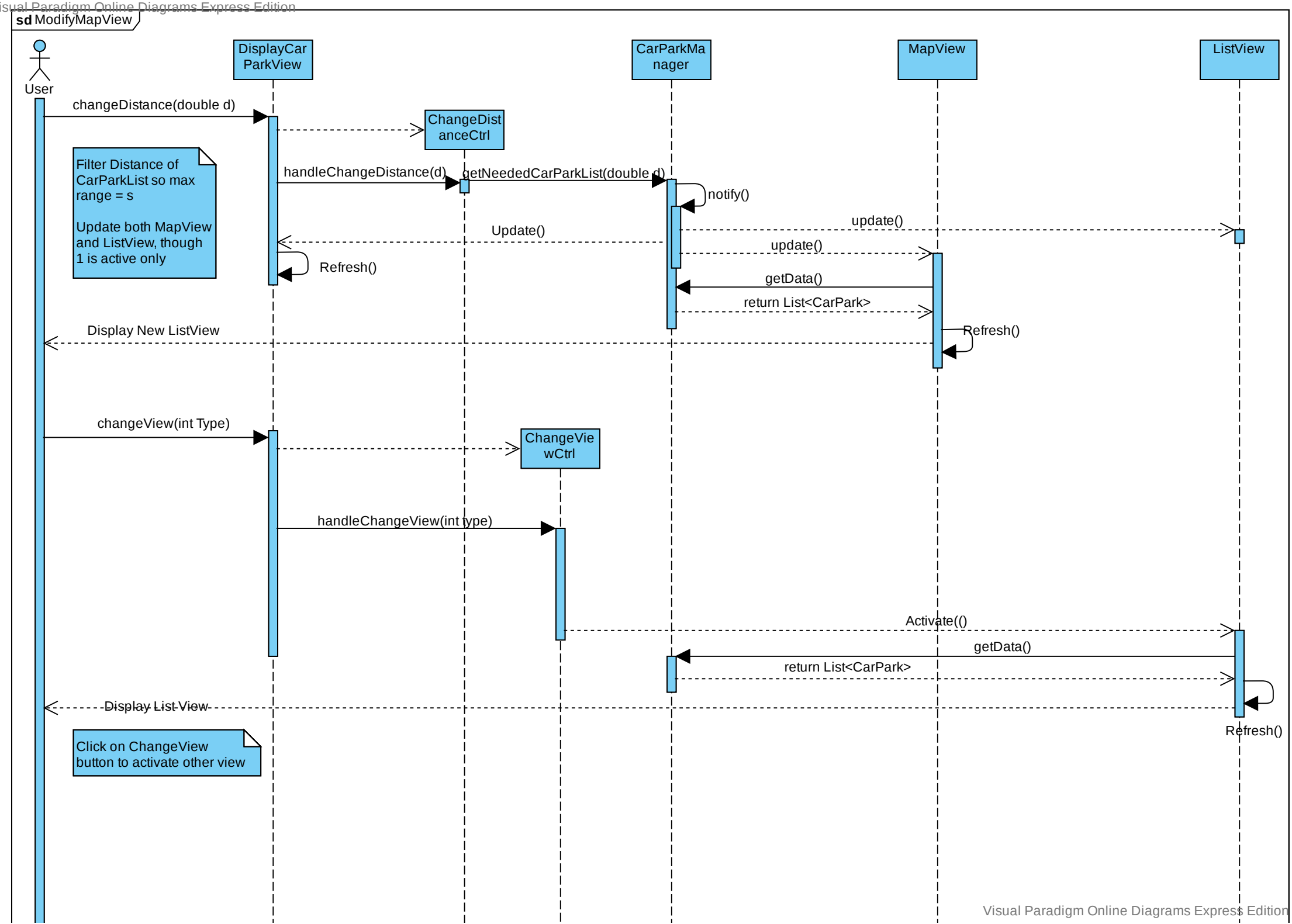
DisplayCarParksView
contain input widgets
to capture users
preferences (View, Distance)

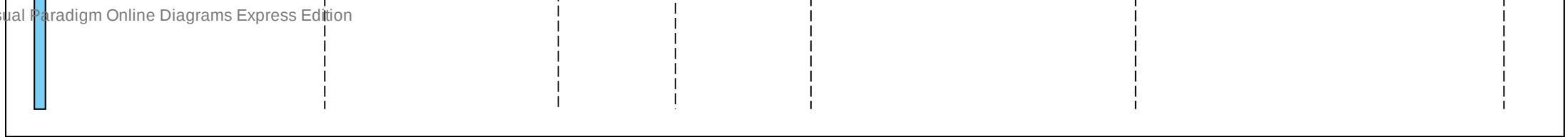
ListView and MapView in charge
of presenting Data and change
presentation based on user preference

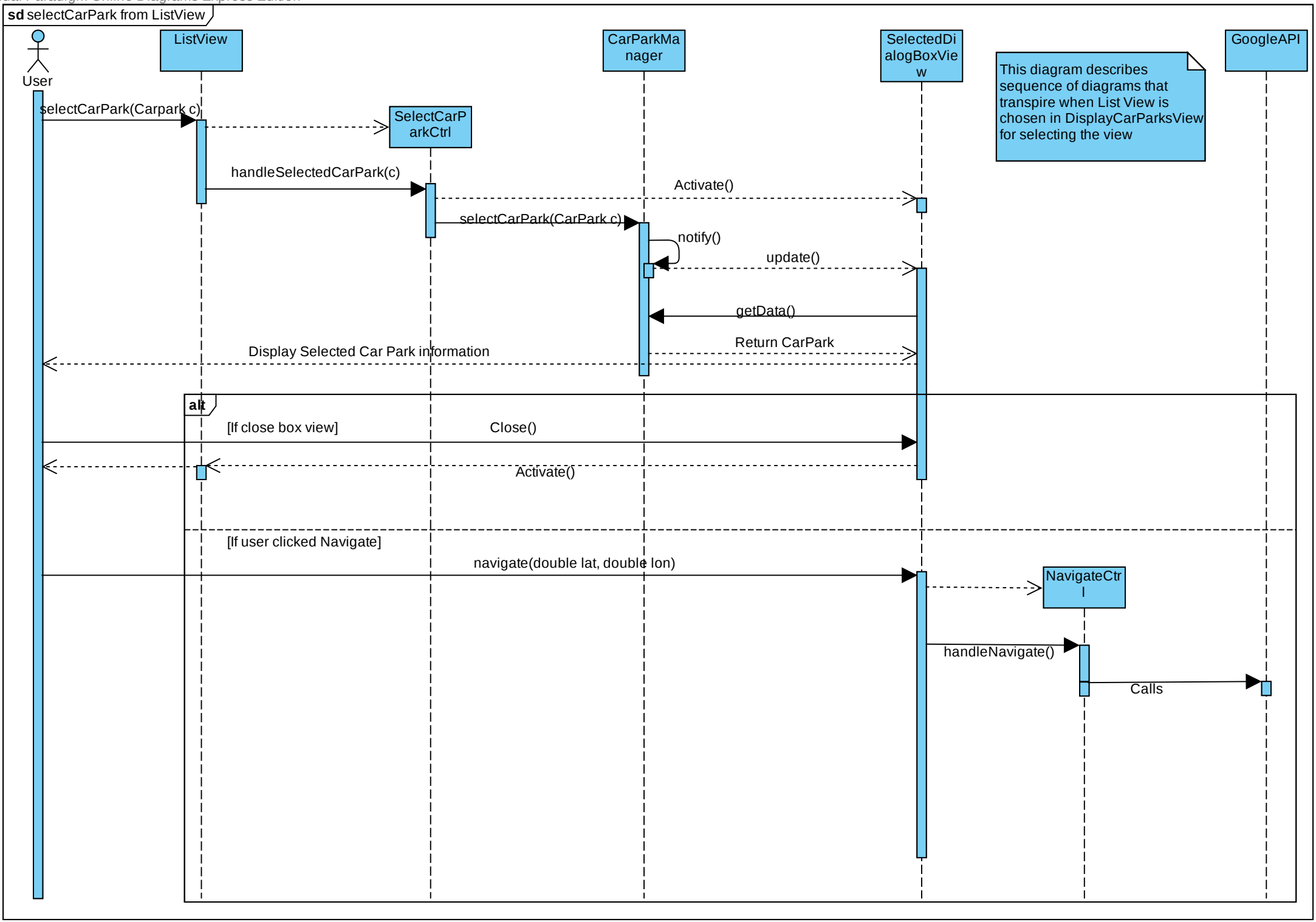
By default : ListView, Distance = 5km,
Sorted by Distance

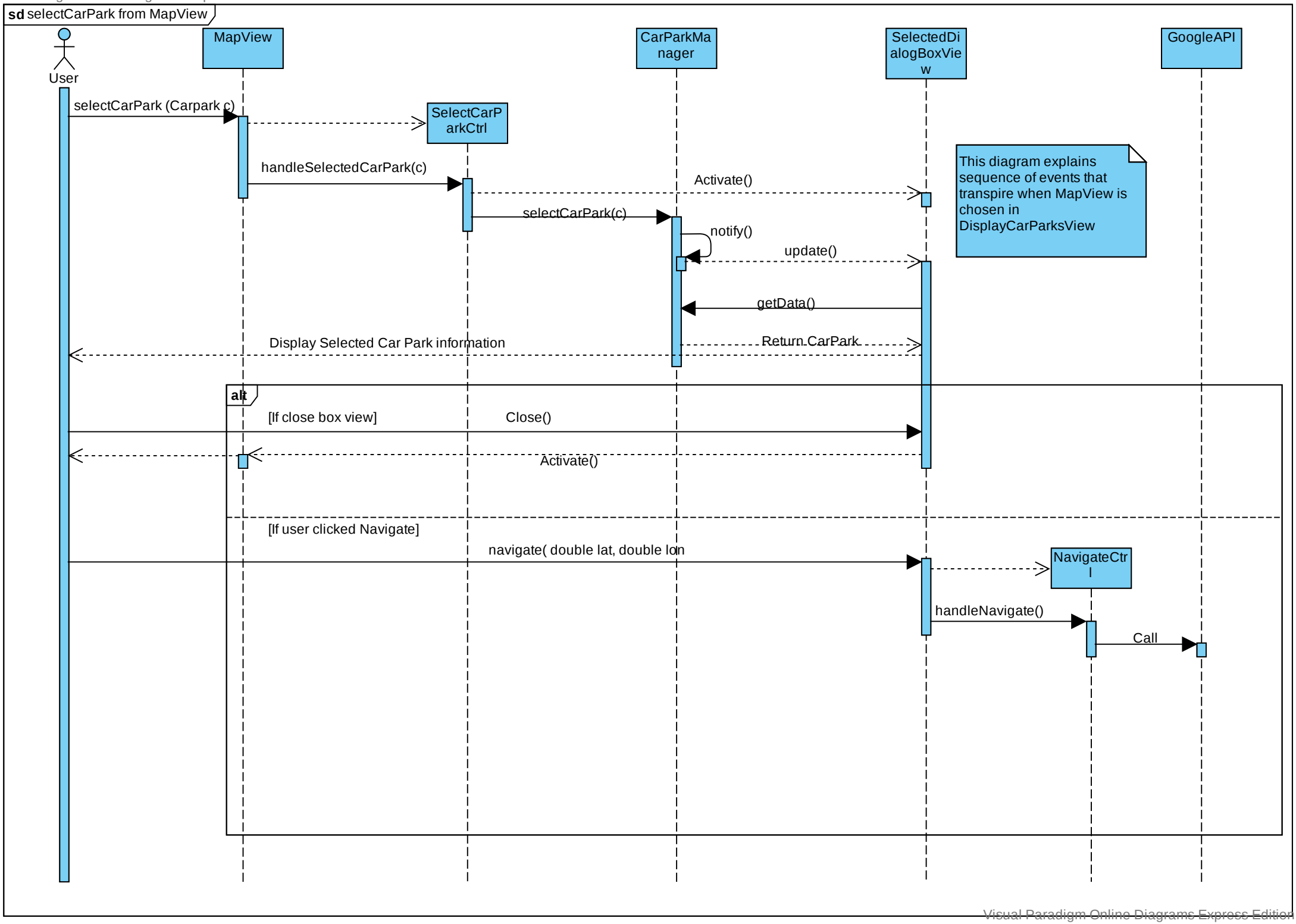


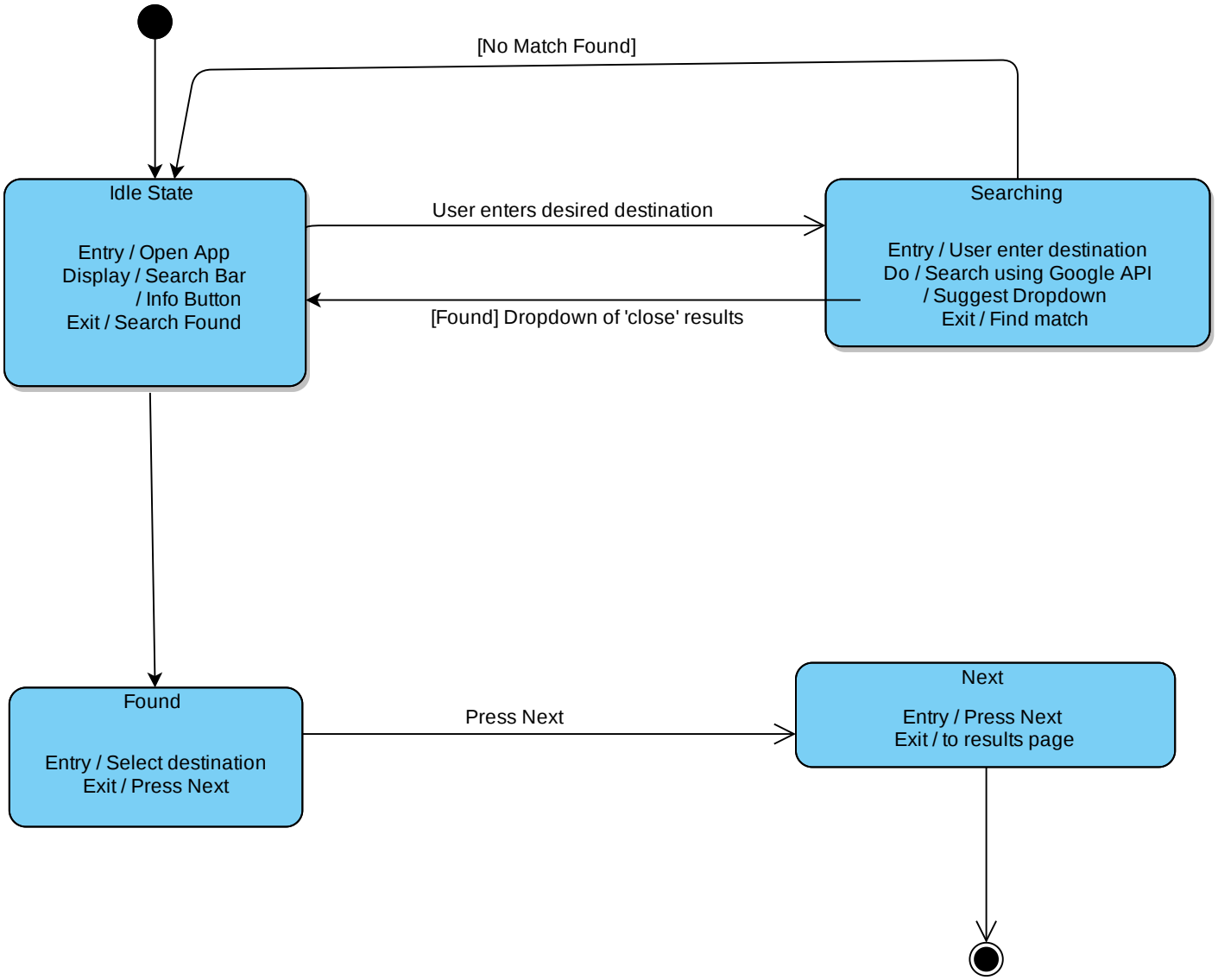


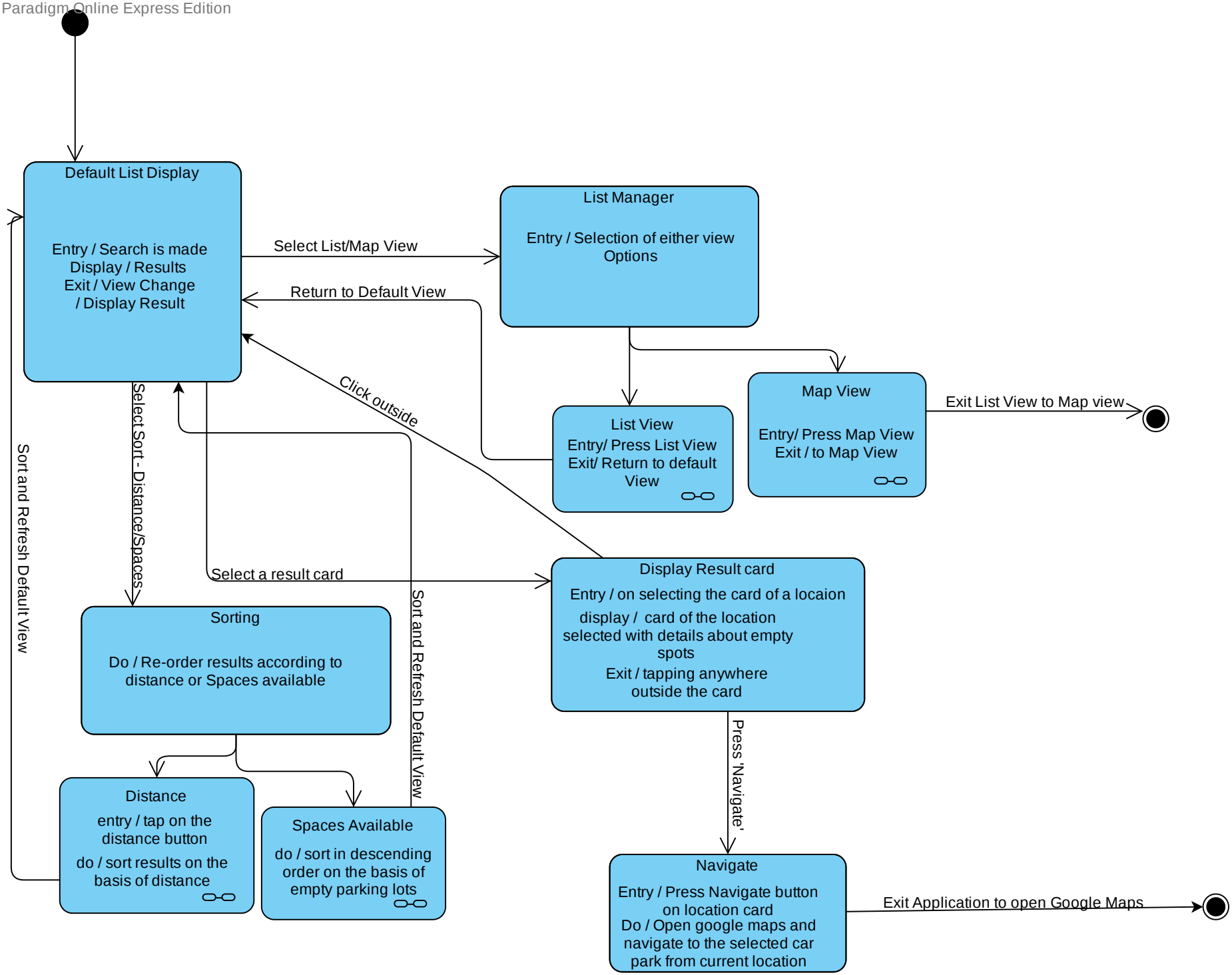


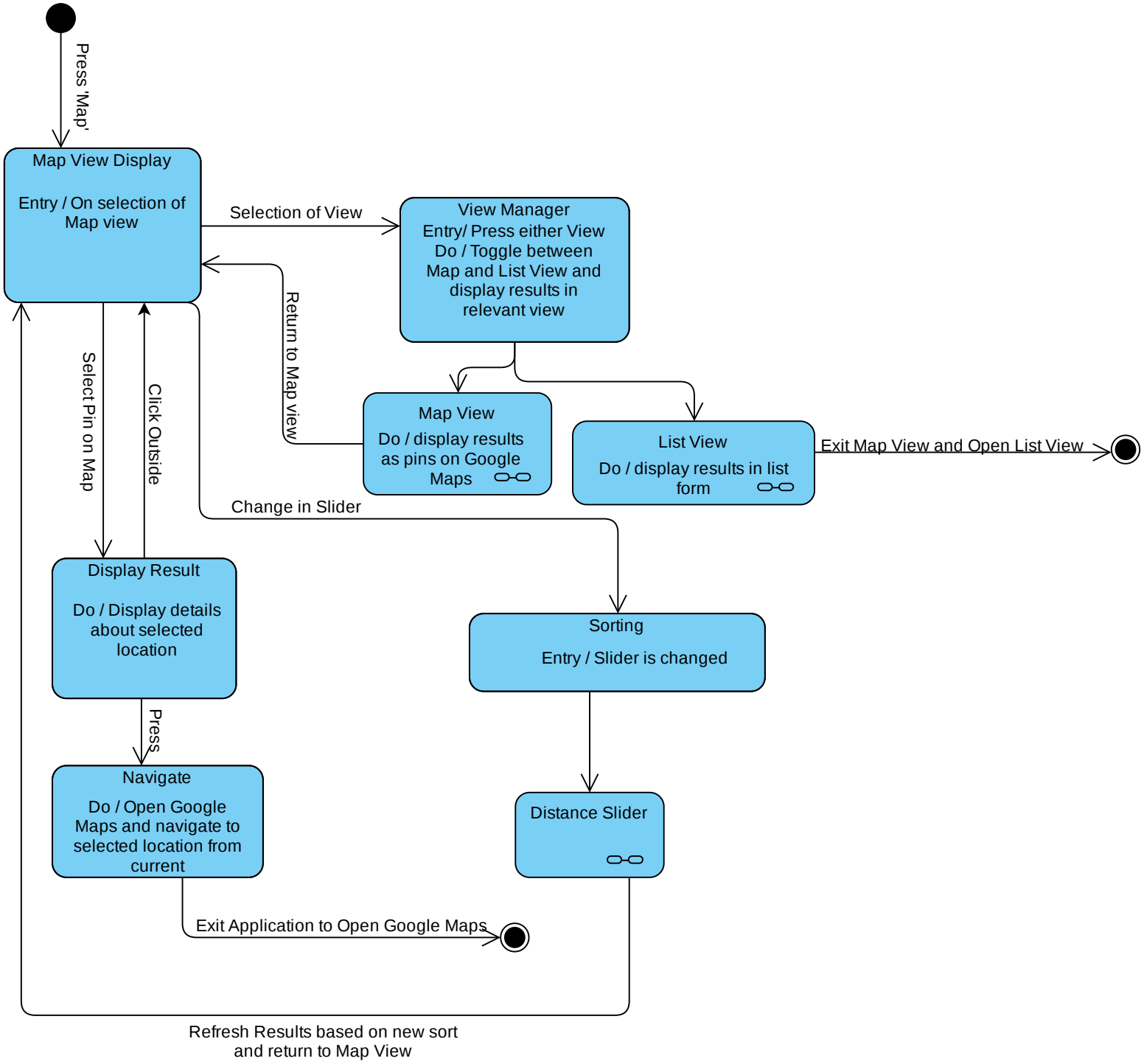












Source:

http://www.frontiernet.net/~kwiegers/process_assets/srs_template.doc