POLITECNICO DI MILANO

Software Engineering 2 Project

myTaxiService

Software Design Document

Simone Guidi

Matteo Imberti

December 4, 2015

Contents

1 Introduction

The Software Design Document is a document to provide documentation which will be used to aid in software development by providing the details of how the software should be built. Within the Software Design Document there is a narrative and graphical documentation of the software design for the project including sequence diagrams, object behavior models, and other supporting requirement information.

1.1 Purpose

The purpose of the Software Design Document is to provide a description of the design of a system detailed enough to allow software developers to proceed with an understanding of what it is to be built and how it is expected to be built. The Software Design Document provides the information necessary to describe the details of the software and system to be built.

1.2 Scope

This Software Design Document is meant to provide a description of a base level system which will work as a proof of concept for the core functionality of the myTaxiService system. This Software Design is focused on the base level system and critical parts of the system.

1.3 Reference Documents

- Assignments 1 and 2 (RASD and DD).pdf
- Structure of the design document.pdf
- RASD.pdf

1.4 Document Structure

This Design Document is divided in four sections:

1. Introduction: gives basic information about the structure, purpose, references of this document.
2. Architectural Design: explains the architectural design choices and description by mean of narrative and UML documentation.
3. Algorithm Design: presents the main interfaces of the system and their use through pseudocode.
4. Requirement Traceability: explains how the requirements we have defined in the RASD map into the design elements that we have defined in this document.

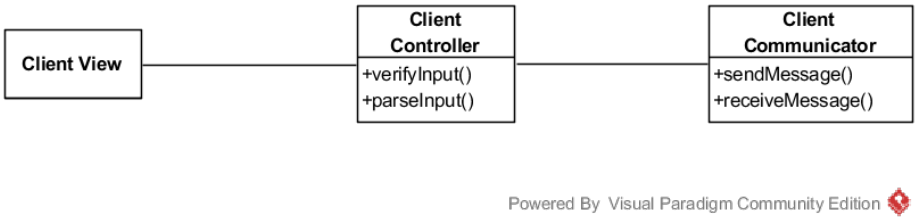
2 Architectural Design

2.1 Overview

The Architectural Design section introduces and gives a brief overview of the design. The System Architecture is a way to give the overall view of a system and to place it into context with external systems. This allows for the readers and users of the document to orientate themselves in the design and see a summary before proceeding into the details of the design.

2.2 High Level Components And Their Interaction

2.2.1 Client Components



Cilent View

- Sends input to the controller for further elaboration.

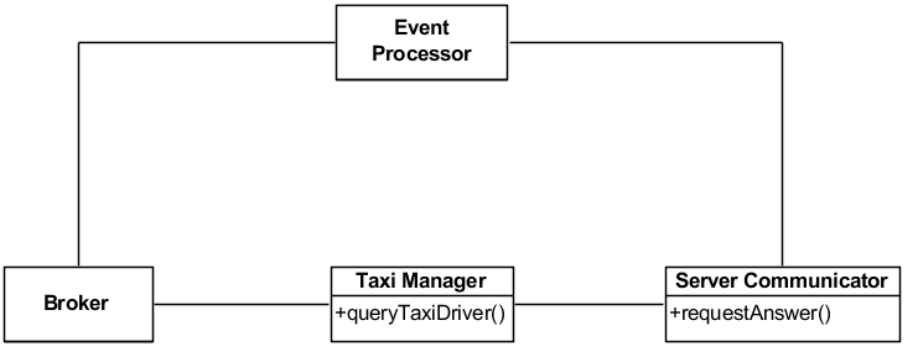
Client Controller

- Receives and sends messages from/to the Client Communicator and handles user interaction with the Client View.

Client Communicator

- Sends and receive messages over the network in both asynchronous and synchronous ways.

2.2.2 Server Components



Event Processor

- Receives messages from the Server Communicator and dispatches the operation associated with the events to the correct Broker. Sends message to clients using the Server Communicator.

Broker

- Performs event handling and Taxi Driver interrogation through the Taxi Manager if necessary.

Taxi Manager

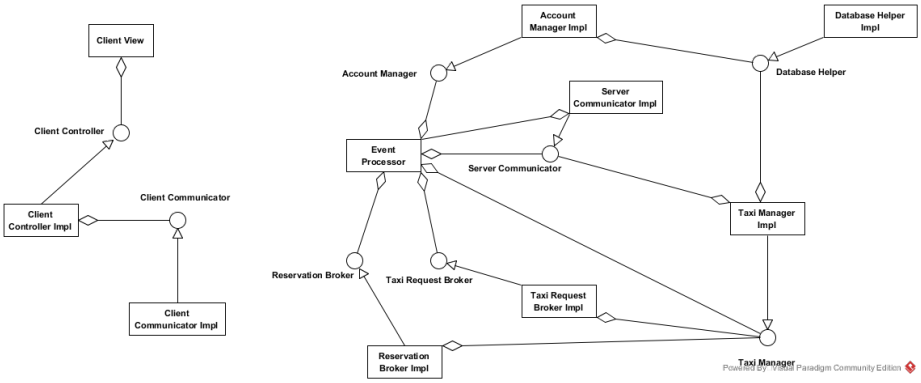
- Queries Taxi Driver with synchronous request/response messages by mean of Server Communicator.

Server Communicator

- Sends and receives messages to/from clients over the network in both synchronous and asynchronous ways.

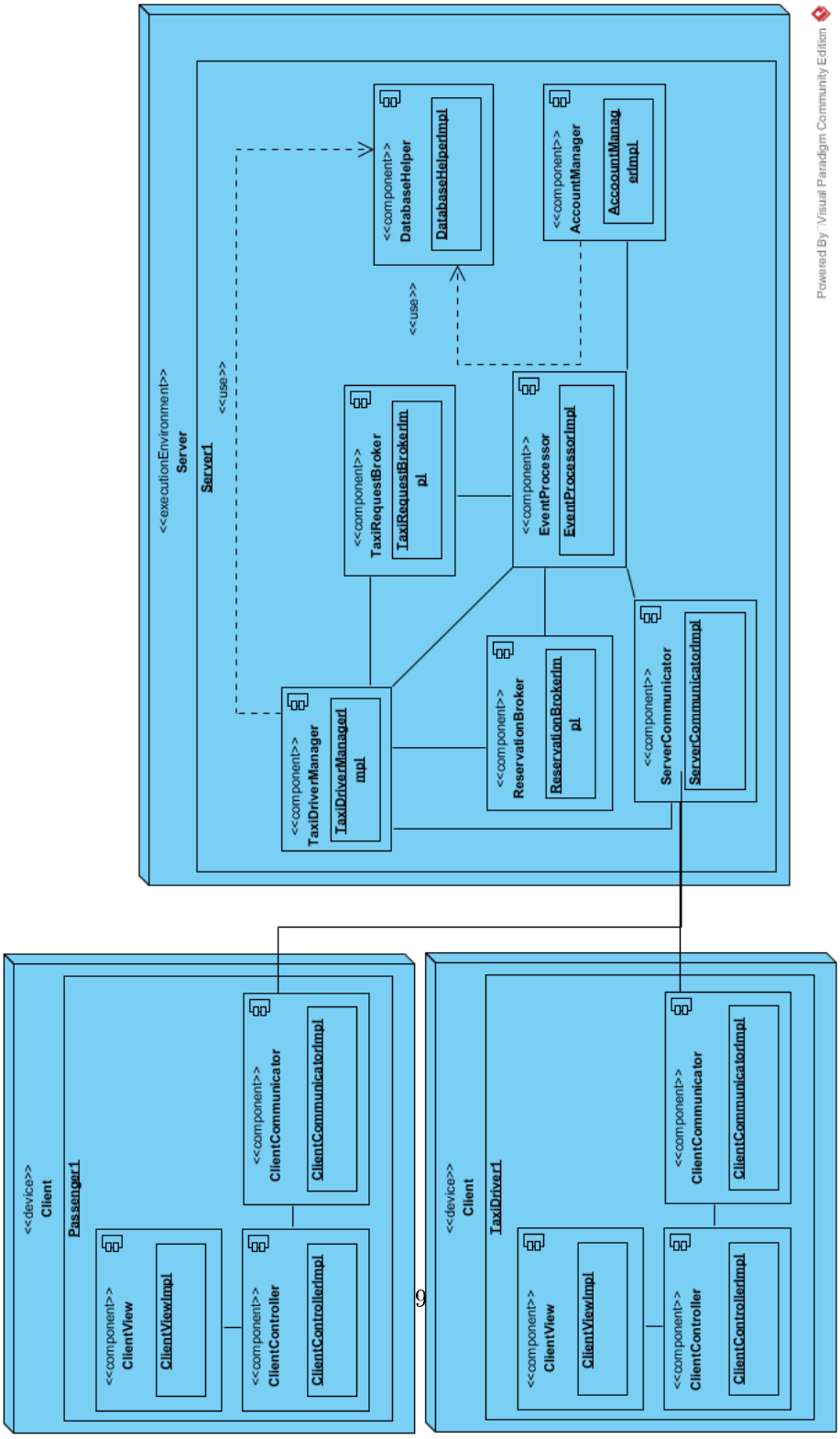
2.3 Component View

This UML schema represents a view of the component and their interfaces.



2.4 Deployment View

In the example below two kind of clients logic are shown: TaxiDriver and Passenger.

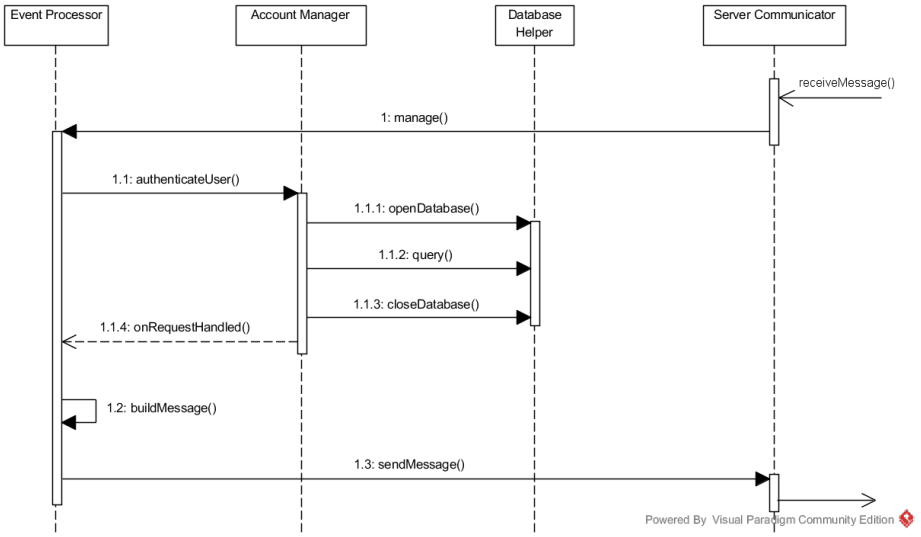


2.5 Runtime View

In these sections sequence diagrams are used to describe the way components interact to accomplish specific tasks typically related to the use cases.

2.5.1 Account Related Operations

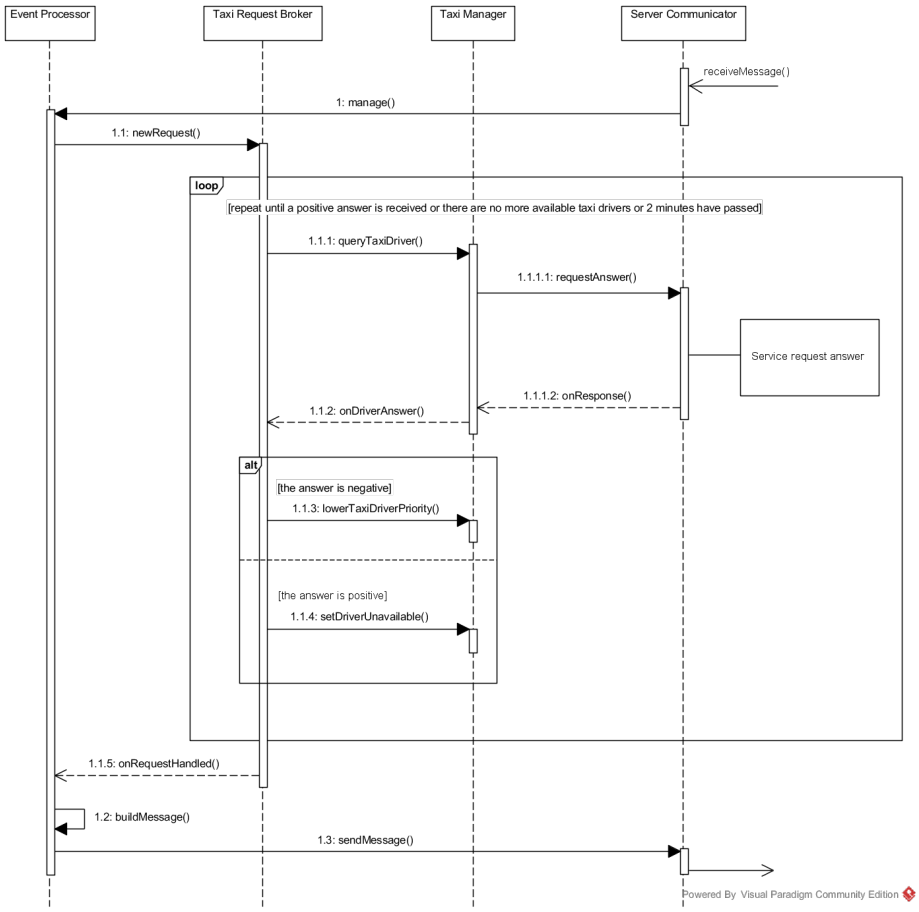
Related Use Cases: ***RASD.3.3.1***, ***RASD.3.3.2***, ***RASD.3.3.3***, ***RASD.3.3.4***, ***RASD.3.3.8***



In this example it is shown the sequence for the login use case, but no significant differences can be found in other account related operations.

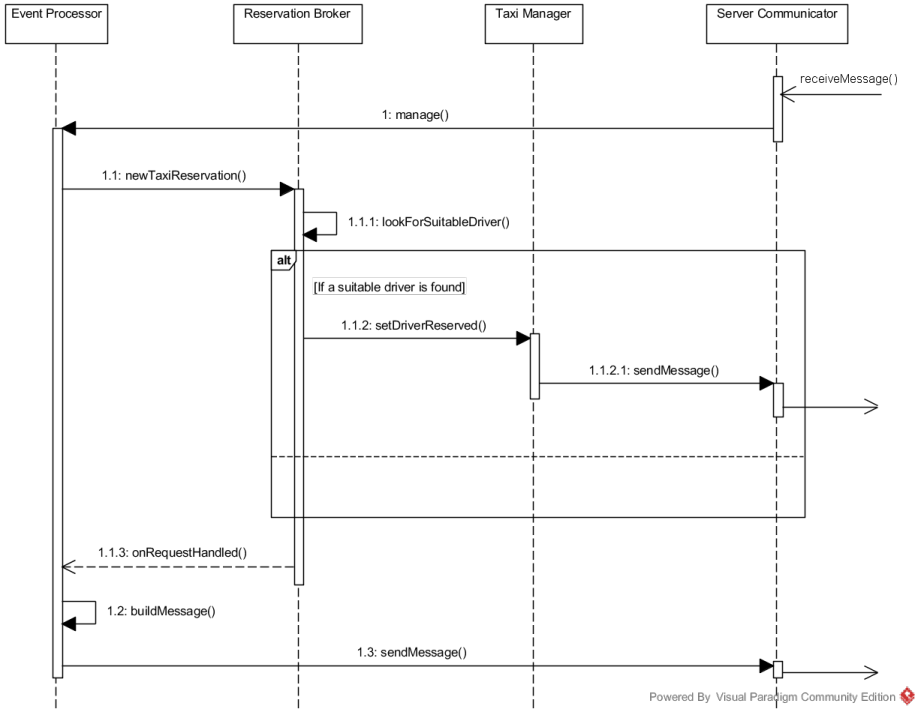
2.5.2 Taxi Request Handling

Related Use Cases: ***RASD.3.3.5, RASD.3.3.6, RASD.3.3.7***



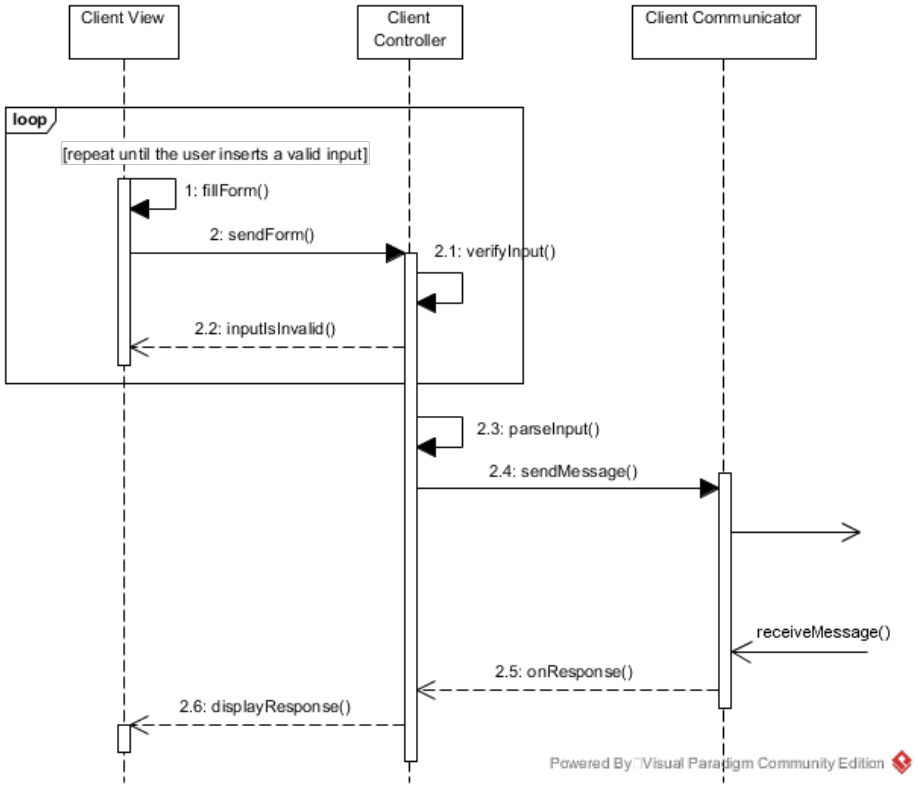
2.5.3 Reservation Request Handling

Related Use Cases: ***RASD.3.3.9, RASD.3.3.10***



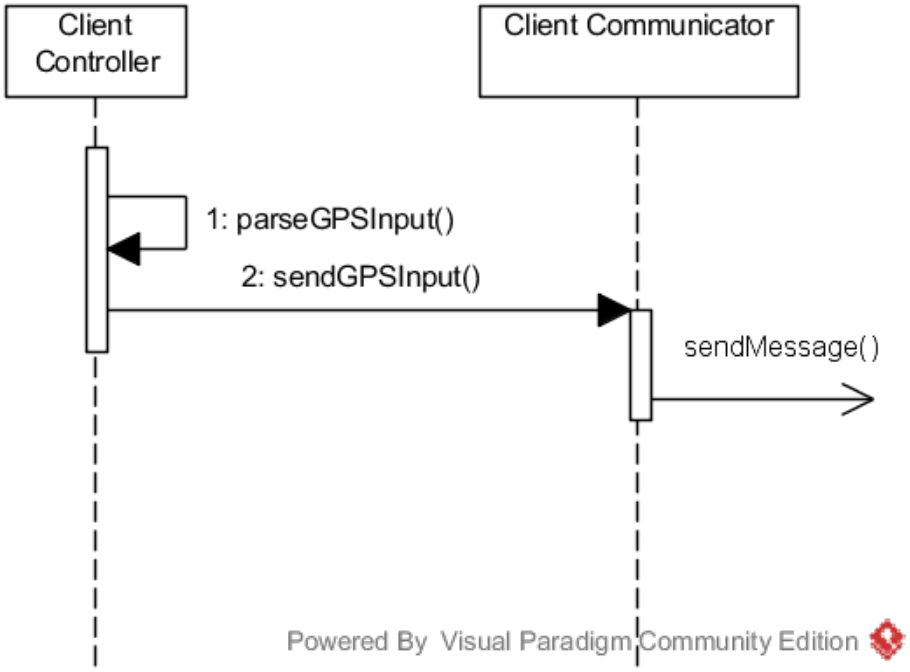
2.5.4 Client Request

Related Use Cases: ***RASD.3.3.1***, ***RASD.3.3.2***, ***RASD.3.3.3***, ***RASD.3.3.4***, ***RASD.3.3.5***, ***RASD.3.3.11***, ***RASD.3.3.12***



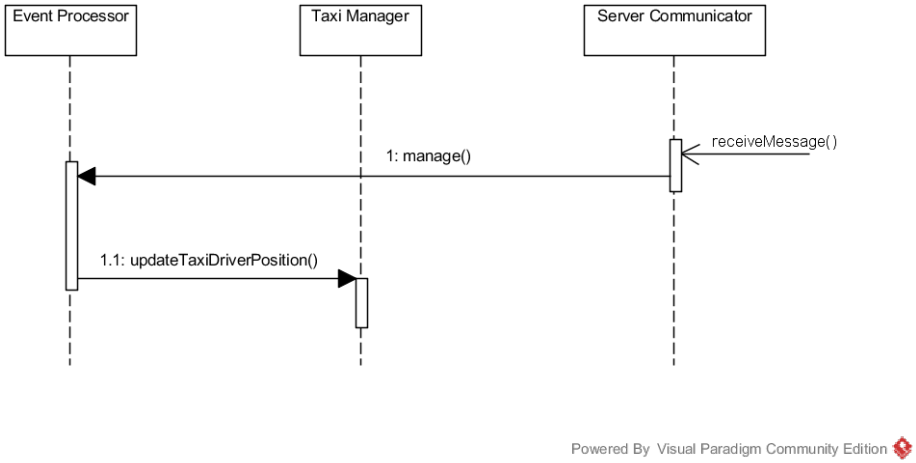
2.5.5 Client GPS Update

Related Use Cases: **system related**



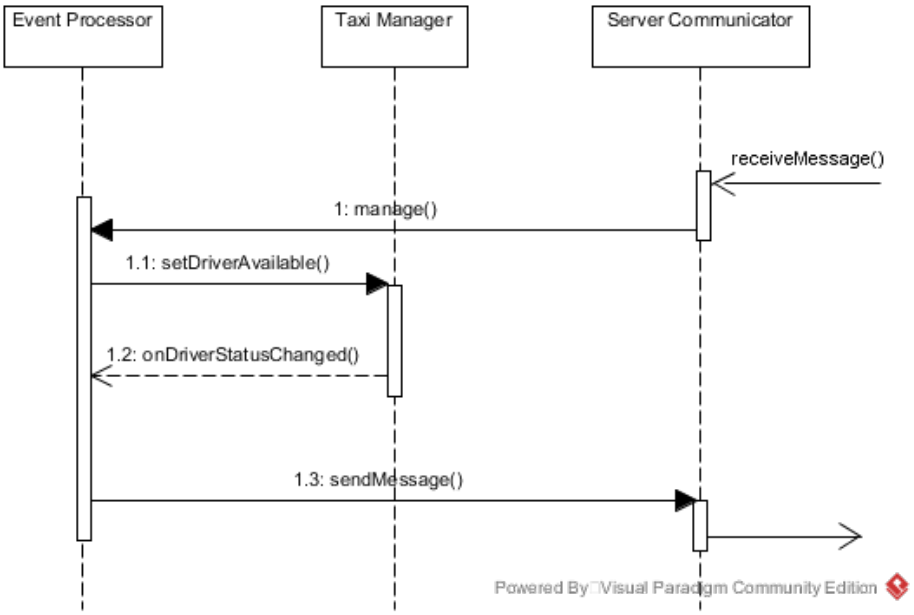
2.5.6 Server GPS Update

Related Use Cases: **system related**



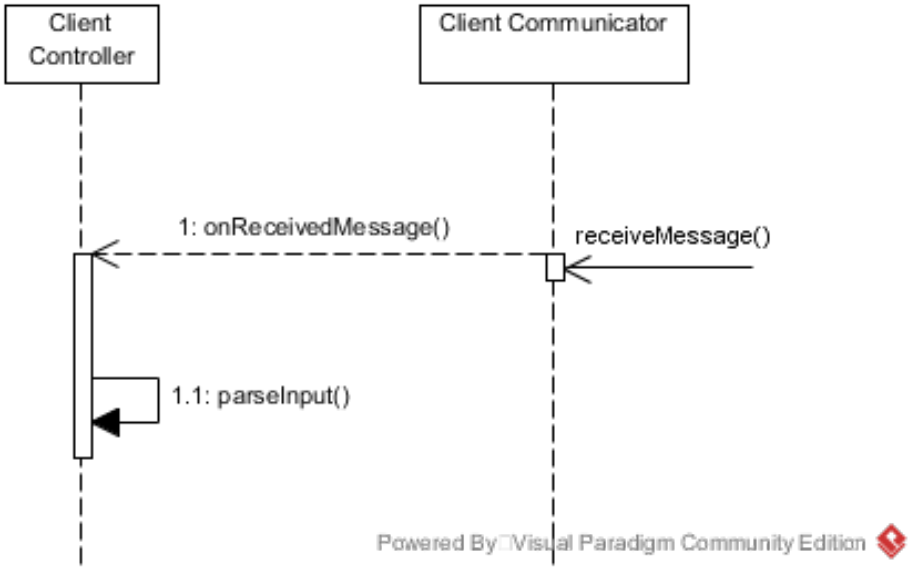
2.5.7 Driver Availability Change

Related Use Cases: ***RASD.3.3.11, RASD.3.3.12***



2.5.8 Client Incoming Message

Releated Use Cases: **system related**



2.6 Component Interfaces

In this section the main interfaces of the system are described in pseudocode (with Java-like syntax).

2.6.1 TaxiManager

```
1  interface TaxiManager
2  {
3      // returns the Zone that contains the specified
           position or null if no Zone contains it
4      void getZone(GPSCoordinate position);
5
6          // returns true if the TaxiDriver accepts, false
               otherwise
7          boolean queryTaxiDriver(TaxiDriver taxiDriver);
8
9          // sets the taxiDriver status to AVAILABLE for
               availabilityTime. If taxiDriver status is not
10         // UNAVAILABLE WrongDriverStatusException is
               thrown
11         void setDriverAvailable(TaxiDriver taxiDriver,
               long availabilityTime) throws
               WrongDriverStatusException;
12
13         // sets the taxiDriver status to UNAVAILABLE. If
               taxiDriver status is RESERVED
14         // WrongDriverStatusException is thrown
15         void setDriverUnavailable(TaxiDriver taxiDriver)
               throws WrongDriverStatusException;
16
17         // sets the taxiDriver status to RESERVED. if
               taxiDriver status is not AVAILABLE
18         // WrongDriverStatusException is thrown
19         void setDriverReserved(TaxiDriver taxiDriver,
               Date time) throws WrongDriverStatusException;
20
21         // sets the taxiDriver priority to minimum
22         void lowerTaxiDriverPriority(TaxiDriver
               taxiDriver);
23
24         // updates the taxiDriver position in the system
               and if the new position belong to a
25         // different TaxiZone, taxiDriver is removed from
                his queue and inserted into the corresponding
```

```
26          // queue of the new TaxiZone at the lowest
                priority
27          void updateTaxiDriverPosition(TaxiDriver
                taxiDriver, GPSCoordinate newPostion);
28  }
```

2.6.2 ReservationBroker

```
1   interface ReservationBroker
2   {
3           // processes the creation of a taxi reservation
                request.
4           // if the request can be created return a
                TaxiReservationInfo, else a
                TaxiReservationException is thrown
5           TaxiDriver newTaxiReservation(GPSCoordinate
                position, Date time) throws
                TaxiReservationException;
6   }
```

2.6.3 DatabaseHelper

```
1   interface DatabaseHelper
2   {
3           // opens a connection to database
4           void openDatabase(Database database);
5
6           // returns a Cursor for the result set of the
                sqlQuery on the database. If the sqlQuery is
7           // not a vald SQL query a SQLException is thrown
8           Cursor query(String sqlQuery) throws SQLException
                ;
9
10          // closes the connection to open databases
11          void closeDatabase();
12  }
```

2.6.4 AccountManager

```
1   interface AccountManager
2   {
3           // authenticates the user in the system, if
                somthing went
```

```
4           // wrong an AuthenticationException is thrown
5           void authenticateUser(AccountInfo accountInfo)
                throws AuthenticationException;
6
7           // registers a user profile in the system, if
                something
8           // went wrong a SignupException is thrown
9           void signup(RegistrationInfo registrationInfo)
                throws SignupException;
10
11          // changes the password for a specified user
12          void changePassword(AccountInfo accountInfo,
                String newPassword);
13
14          // deauthenticates the user from the system
15          void logout(AccountInfo accountInfo);
16  }
```

2.6.5 ClientCommunicator

```
1   interface ClientCommunicator
2   {
3           // Sets the stream for communication
4           void openStream(Socket socket);
5
6           // Writes a message on the outputStream
7           void sendMessage(String message);
8
9           // Read a message from the inputStream
10          String receiveMessage();
11  }
```

2.6.6 ServerCommunicator

```
1   interface ServerCommunicator
2   {
3           // Sets the stream for communication
4           void openStream(Socket socket);
5
6           // Writes a message on the outputStream
7           void sendMessage(String message);
8
9           // Read a message from the inputStream
10          String receiveMessage();
```

```
11
12          // Waits for a synchronous answer
13          boolean requestAnswer(String message);
14  }
```

2.6.7 TaxiRequestBroker

```
1   interface TaxiRequestBroker
2   {
3           // processes the creation of a taxi request.
4           // if the request can be created return a
                TaxiRequestInfo, else a TaxiRequestException
                is thrown
5           TaxiDriver newTaxiRequest(GPSCoordinate position)
                throws TaxiRequestException;
6   }
```

2.7 Architectural Styles and Patterns

The system myTaxiService will use a Service-oriented architecture. The business logic of the system is distributed through dedicated application servers in the middle layer of a 3-layer architecture. The presentation layer is located on the server side.
Pattern used on the client side:

- MVC: The client has no intensive business logic and the controller is responsable only for input checking, interaction with the view and information gathering and providing from the model through a communication channel over the web. This pattern allows to support different client kinds with little changes for each one.

Pattern used on the server side:

- Broker: The server needs to perform various operation on queues before targetting the right Taxi Driver user with a message.
- Request/Response: Taxi Drivers interrogation often requires a synchronous message exchange between server and clients (i.e. Taxi Request Answer) in which the server requests the client for an immediate response.

3 Algorithm Design

3.1 TaxiManagerImpl

```
1
2  public Zone getZone(GPSCoordinate position)
3  {
4    for(Zone zone : this.taxiZones)
5      if(zone.isCoordinateWithin(taxiCoordinate))
6            return zone;
7    return null;
8  }
9
10 private Zone getTaxiDriverZone(TaxiDriver taxiDriver)
11 {
12   return getZone(taxiDriver.getTaxi().getCurrentPosition
         ());
13 }
14
15 public boolean queryTaxiDriver(TaxiDriver taxiDriver)
16 {
17   return this.serverCommunicator.requestAnswer(JSON);
18 }
19
20 public void setDriverAvailable(TaxiDriver taxiDriver,
     Date availabilityTime) throws
     WrongDriverStatusException
21 {
22   if(!taxiDriver.getStatus().equals(Status.UNAVAILABLE))
23     throw new WrongDriverStatusException();
24   taxiDriver.setStatus(Status.AVAILABLE);
25   taxiDriver.setAvailabilityTime(availabilityTime);
26   Zone zone = getTaxiDriverZone(taxiDriver);
27   zone.getQueue().addTaxiDriver(taxiDriver);
28 }
29
30 public void setDriverUnavailable(TaxiDriver taxiDriver)
     throws WrongDriverStatusException
31 {
32   if(!taxiDriver.getStatus().equals(Status.AVAILABLE))
33     throw new WrongDriverStatusException();
34   Zone zone = getTaxiDriverZone(taxiDriver);
35   zone.getQueue().removeTaxiDriver(taxiDriver);
36   taxiDriver.setStatus(Status.UNAVAILABLE);
37   taxiDriver.setAvailabilityTime(null);
38 }
39
40 public void setDriverReserved(TaxiDriver taxiDriver)
     throws WrongDriverStatusException
```

```
41 {
42   if(!taxiDriver.getStatus().equals(Status.AVAILABLE))
43     throw new WrongDriverStatusException();
44   taxiDriver.setStatus(Status.RESERVED);
45 }
46
47 public void lowerTaxiDriverPriority(TaxiDriver taxiDriver
     )
48 {
49   Zone zone = getTaxiDriverZone(taxiDriver);
50   if(zone != null)
51     zone.getQueue().shiftTaxiDriverToTail(taxiDriver);
52 }
53
54 public void updateTaxiDriverPosition(TaxiDriver
     taxiDriver, GPSCoordinate newPosition)
55 {
56   if(taxiDriver.getStatus().equals(Status.UNAVAILABLE))
57     return;
58
59   Zone oldZone = getTaxiDriverZone(taxiDriver);
60   taxiDriver.getTaxi().setCurrentPosition(newPosition);
61   Zone newZone = getTaxiDriverZone(taxiDriver);
62
63   if(oldZone != newZone)
64   {
65     oldZone.getQueue().removeTaxiDriver(taxiDriver);
66          if(newZone == null)
67            setDriverUnavailable(taxiDriver);
68          else
69            newZone.getQueue().addTaxiDriver(taxiDriver);
70   }
71 }
```

3.2 TaxiRequestBrokerImpl

```
1  public TaxiDriver newTaxiRequest(GPSCoordinate position)
       throws TaxiRequestException{
2    Zone zone = this.taxiManager.getZone(position);
3    TaxiDriver head;
4    TaxiDriver replacement = null;
5      while(!TIMEOUT)
6      {
7        head = zone.getQueue().getHead();
8            while(!TIMEOUT)
```

```
9         {
10          if (head == null)
11            throw new TaxiRequestException();
12            if (head.getStatus() == RESERVED)
13            {
14              replacement = lookForReservableDriver(
                    position, head.getReservation().
                    getScheduledTime());
15              if (replacement == null)
16              {
17                head = zone.getQueue().getHeadAfter(head);
18              }
19              else break;
20            }
21        }
22        if (TIMEOUT)
23          break;
24        if(!this.taxiManager.queryTaxiDriver(head))
25        {
26          this.taxiManager.lowerTaxiDriverPriority(head);
27        }
28        else
29        {
30          if (replacement != null)
31            this.taxiManager.setDriverReserved(position,
                  head.getReservation().getScheduledTime())
32          this.taxiManager.setDriverUnavailable(head);
33          return head;
34        }
35      }
36      if(TIMEOUT){
37        throw new TaxiRequestException();
38      }
39  }
40
41
42
43
44  private TaxiDriver lookForReservableDriver(GPSCoordinate
      position, Date time) throws TaxiRequestException{
45    Zone reservationZone = this.taxiManager.getZone(
        position);
46    if (reservationZone == null)
47      throw new TaxiRequestException();
48    TaxiDriver driver = reservationZone.
        getFirstAvailableTaxi(time):
```

```
49    return driver;
50  }
```

3.3 ReservationBrokerImpl

```
1   public TaxiDriver newTaxiReservation(GPSCoordinate
        position, Date time) throws TaxiReservationException{
2     TaxiDriver driver = lookForSuitableDriver(position,
          time);
3     this.taxiManager.setDriverReserved(driver,time);
4     return driver;
5   }
6
7   private TaxiDriver lookForSuitableDriver(GPSCoordinate
        position, Date time) throws TaxiReservationException{
8     Zone reservationZone = this.taxiManager.getZone(
          position);
9     if (reservationZone == null)
10      throw new TaxiReservationException();
11    TaxiDriver driver = reservationZone.
          getFirstAvailableTaxi(time):
12    if (driver == null)
13    {
14      for (Zone z : reservationZone.getNeighbours())
15      {
16        driver = z.getQueue().getFirstAvailableTaxi(time);
17        if (driver != null)
18          break;
19      }
20      if (driver == null)
21      {
22        throw new TaxiReservationException();
23      }
24    }
25    return driver;
26  }
```

4 Requirement Traceability

The functional requirements listed in this section refers to the RASD.pdf document section **3.2.**
Each functional requirement is associated to a set of components which relize the funcion.

4.1 R1

Components: ***ClientController, EventProcessor, TaxiRequestBroker, TaxiManager***

4.2 R2

Components: ***ClientController, ReservationBroker, EventProcessor, TaxiManager***

4.3 R3

Components: ***ClientController, ReservationBroker***

4.4 R4

Components: ***ClientController, EventProcessor, TaxiManager***

4.5 R5

Components: ***ClientController, TaxiManager, EventProcessor***

4.6 R6

Components: ***EventProcessor, ReservationBroker, TaxiManager, TaxiRequestBroker***

4.7 R7

Components: ***ClientController, AccountManager, EventProcessor***

4.8 R8

Components: ***ClientController, EventProcessor, AccountManager***

4.9 R9

Components: ***ClientController, EventProcessor, AccountManager***

4.10 R10

Components: ***EventProcessor, ReseravationBroker***