

# **Cab Dispatching System**

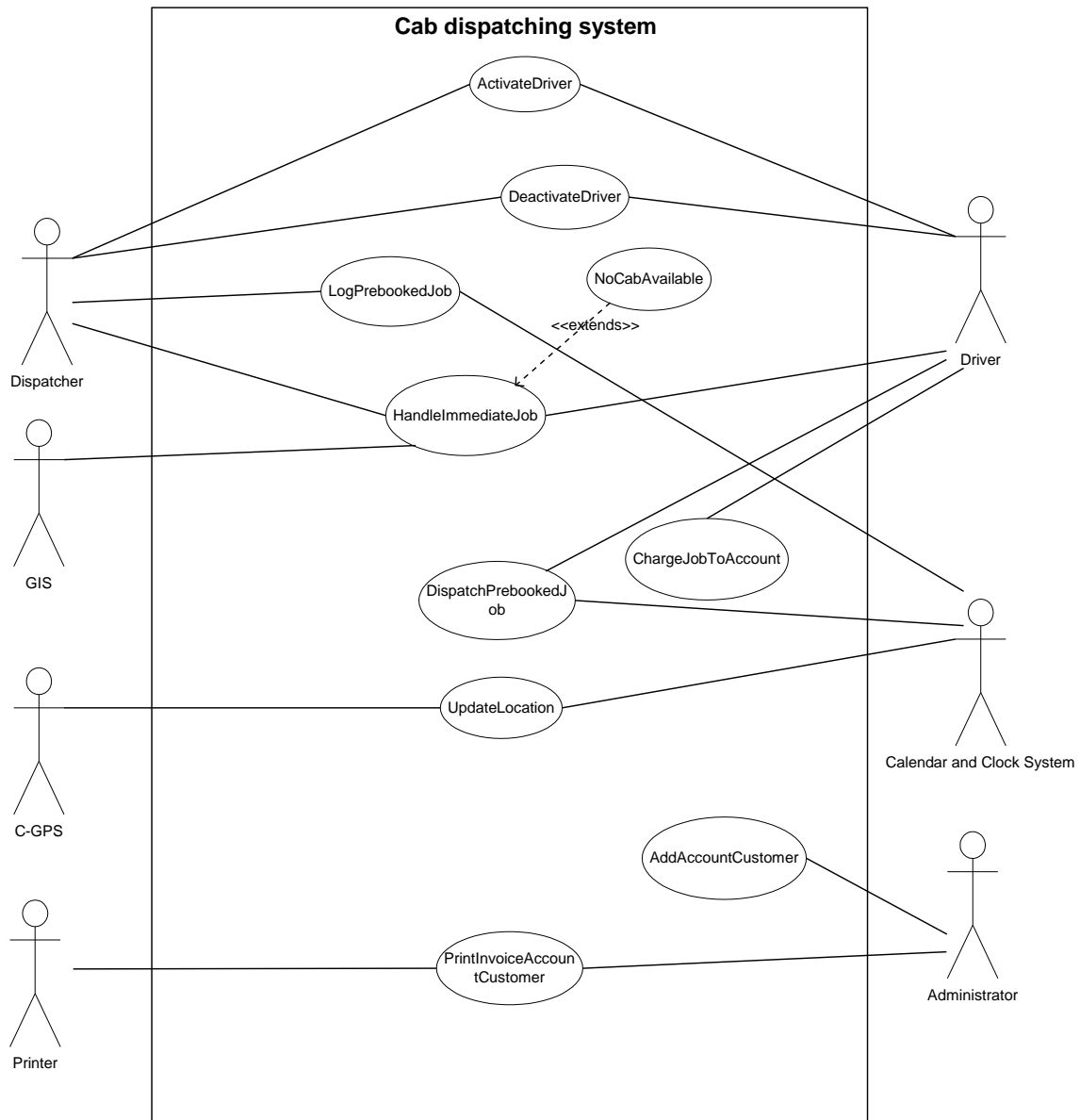
# 1. Summary

1.	Summary .....	2
2.	Use Cases .....	4
2.1.	Use Case Diagram.....	4
2.2.	Actors .....	5
2.3.	Use Case Descriptions .....	6
2.3.1	Activate Driver Use Case Description .....	6
2.3.2	Deactivate Driver Use Case Description .....	6
2.3.3	Update Location Use Case Description .....	7
2.3.4	Handle Immediate Job Use Case Description.....	7
2.3.5	No Cab Available Use Case Description .....	8
2.3.6	Log Pre-booked Job Use Case Description .....	8
2.3.7	Dispatch Pre-booked Job Use Case Description.....	9
2.3.8	Add Account Customer Use Case Description .....	9
2.3.9	Charge Job To Account Use Case Description.....	10
2.3.10	Print Invoice Account Customer Use Case Description .....	10
3.	Class Diagram .....	11
4.	Data Dictionary .....	13
4.1.	Entity classes.....	13
4.1.1	Cab .....	13
4.1.2	Customer .....	17
4.1.3	Driver .....	18
4.1.4	Job .....	19
4.1.5	ImmediateJob.....	19
4.1.6	PrebookedJob.....	21
4.2.	Control Classes .....	23
4.2.1	CustomerControl.....	23
4.2.2	DriverControl .....	24
4.2.3	JobManagerControl.....	25
4.2.4	UpdateLocationControl.....	25
4.3.	Boundary Classes .....	26
4.3.1	AddAccountForm .....	26
4.3.2	ChargeAccountForm.....	26
4.3.3	DispatchJobForm .....	27
4.3.4	InvoiceAccountCustomer .....	27
4.3.5	ReportCabLocation .....	28
4.3.6	ChangeDriverStatusForm .....	28
4.3.7	RequestCabForm.....	29
4.3.8	NotifyNewJob .....	30
4.3.9	ReportCabETA .....	30
4.3.10	ClockBoundry .....	30
4.4.	Associations between entity classes.....	32
5.	Sequence diagrams.....	34
5.1.	Abbreviations .....	34
5.2.	HandleImmediateJob Sequence Diagram .....	34

5.3.	LogPrebookedJob Sequence Diagram .....	35
5.4.	DispatchPrebookedJob Sequence Diagram .....	36
6.	State charts .....	37
6.1.	State chart for an object of class Job.....	37
6.2.	Statechart for an object of class Cab.....	38

## 2. Use Cases

### 2.1. Use Case Diagram



## 2.2. Actors

1. **Dispatcher:** receives calls from customers, activates and deactivates drivers, logs pre-booked jobs, and handles immediate job.
2. **Driver:** charges job to account if a customer chooses to do so. A Driver communicates with the system using an onboard computer present on each cab
3. **Calendar and Clock System:** triggers the automatic dispatch of a cab to service a job, returns the current date/time; triggers the cab location to be updated by CGPS
4. **Printer:** prints a customer's account information (i.e., the customer's outstanding bills).
5. **Cab-Global Positioning System (CGPS):** periodically indicates the position of cabs to the system.
6. **Administrator:** administrates and updates the databases (including the account customer database).
7. **Geographical Information System (GIS):** handles geographical information (routes, sites, roads, traffic, etc.) about the cabs activity area. In particular, it allows computing the Estimated Time of Arrival (ETA) from a source position to a target position, given the GPS coordinates of the two positions.

Important Note: The fact that a cab can take a customer without a phone call to the dispatcher (i.e., the customer directly calls the cab on the road) is not considered in this analysis.

## 2.3. Use Case Descriptions

### 2.3.1 Activate Driver Use Case Description

Use Case Name:        `ActivateDriver`

Participating actor(s):   `Initiated by Dispatcher, communicates with Driver`

Entry conditions:        `The Driver reports for a new shift to the Dispatcher.`

Flow of events:        1. `The Dispatcher informs the system of the availability of the Driver.`  
                               2. `The Driver is added to the list of active drivers, and notified.`

Exit condition:        `Driver's addition to the list of active cabs is confirmed by the Dispatcher.`

Special requirements:   `The confirmation delay time <= 2 minutes.`

### 2.3.2 Deactivate Driver Use Case Description

Use Case Name:        `DeactivateDriver`

Participating actor:    `Initiated by Dispatcher, communicates with Driver.`

Entry conditions:        `The Driver reports end of shift to the Dispatcher.`

Flow of events:        1. `The Dispatcher informs the system of the Driver's end of shift.`  
                               2. `The Driver is removed from the list of available drivers, and notified.`

Exit condition:        `Driver's removal from the list of active cabs is confirmed by the Dispatcher.`

Special requirements:   `The confirmation delay time <= 2 minutes.`

### 2.3.3 Update Location Use Case Description

Use Case Name:	UpdateLocation
Participating actor:	Initiated by Calendar and Clock System, communicates with C-GPS
Entry conditions:	Periodically triggered (every five minutes) by the Calendar and Clock System
Flow of events:	<ol style="list-style-type: none"> <li>1. The Calendar and Clock System periodically triggers this use case to update each cab's position in the system</li> <li>2. The C-GPS returns each cab's position and the information in the system is updated</li> </ol>
Exit condition:	The position of each cab is updated
Special requirements:	The current location of the cabs is updated every five minutes.

### 2.3.4 Handle Immediate Job Use Case Description

Use Case Name:	HandleImmediateJob
Participating actor:	Initiated by Dispatcher, communicates with Driver, GIS
Entry conditions:	A phone call from a customer, asking for a cab, to the Dispatcher
Flow of events:	<ol style="list-style-type: none"> <li>1. The Dispatcher asks the system for the nearest cab to the customer's location.</li> <li>2. Given the customer's location, Estimated Time of Arrival (ETA) is calculated by means of GIS.</li> <li>3. The Dispatcher informs the customer of the ETA: if s/he accepts, the Dispatcher confirms the operation.</li> <li>4. The Driver is informed.</li> </ol>
Exit condition:	The job is successfully allocated and the Driver is informed
Special requirements:	The delay between request and confirmation should be < 2 minutes. The customer should not wait for the cab more than 15 minutes.

### 2.3.5 No Cab Available Use Case Description

Use Case Name:	NoCabAvailable (extends relationship) NoCabAvailable extends HandleImmediateJob when all cabs are busy at the time the Dispatcher asks for the nearest cab to the customer
Entry condition	The Dispatcher is asking for a cab, but no cab is available
Flow of events	<ol style="list-style-type: none"> <li>1. The Dispatcher gets information from the system that all cabs are currently busy.</li> <li>2. The Dispatcher informs the customer to wait for a period of time <math>t</math> before calling again, because there is no cab available.</li> </ol>
Exit condition:	
Special requirements:	The time $t$ should be $\leq 10$ minutes.

### 2.3.6 Log Pre-booked Job Use Case Description

Use Case Name:	LogPrebookedJob
Participating actor:	Initiated by Dispatcher, communicates with Calendar and Clock System
Entry conditions:	A call from a customer to pre-book a cab
Flow of events:	<ol style="list-style-type: none"> <li>1. Dispatcher inputs the details of the job: pickup time, pickup location, id of the customer (or his/her phone number if not an account customer)</li> <li>2. The current date and time (i.e., date/time of booking) are recorded (they're obtained from the Calendar and Clock System)</li> <li>3. A cab is booked for this job.</li> </ol>
Exit condition:	Booking confirmed by the Dispatcher to the customer.
Special requirements:	Cancellation of job must be given 24 hours before job execution.



### 2.3.7 Dispatch Pre-booked Job Use Case Description

Use Case Name:	DispatchPrebookedJob
Participating actor:	Initiated by Calendar and Clock System, communicates with Driver
Entry conditions:	1. The Calendar and Clock System activates this use case every minute.
Flow of events:	2. The system search for pre-booked jobs that should be performed 3. The cab Driver is informed to be on stand-by 15 minutes before the rendezvous time, and to proceed to location.
Exit condition:	The Driver confirms to perform the job.
Special requirements:	The stand-by period should not be more than 15 minutes.

### 2.3.8 Add Account Customer Use Case Description

Use Case Name:	AddAccountCustomer
Participating actor:	Initiated by Administrator
Entry conditions:	A customer calls to create an account
Flow of events:	1. The Administrator searches for the customer 2. If the customer does not exist, the Administrator creates a new account for the customer
Exit condition:	The system is successfully updated

### 2.3.9 Charge Job To Account Use Case Description

Use Case Name:	ChargeJobToAccount
Participating actor:	Initiated by Driver
Entry conditions:	A customer requests the Driver that (s)he wants to use his/her account.
Flow of events:	<ol style="list-style-type: none"> <li>1. The system checks if the customer has a valid account</li> <li>2. The system checks if the amount is within the Customer's credit line.</li> <li>3. The system charges the customer account; credit interest is included in the charge.</li> </ol>
Exit condition:	The system confirms end of job.

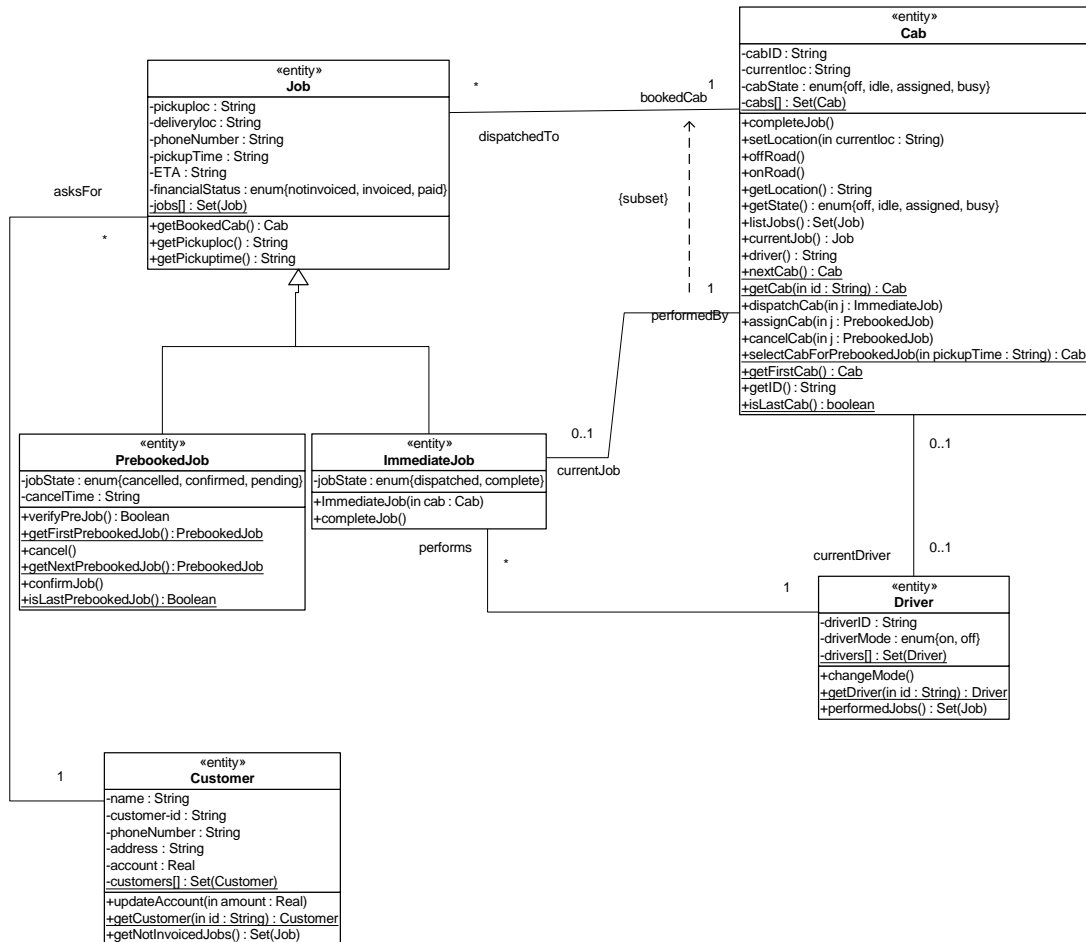
### 2.3.10 Print Invoice Account Customer Use Case Description

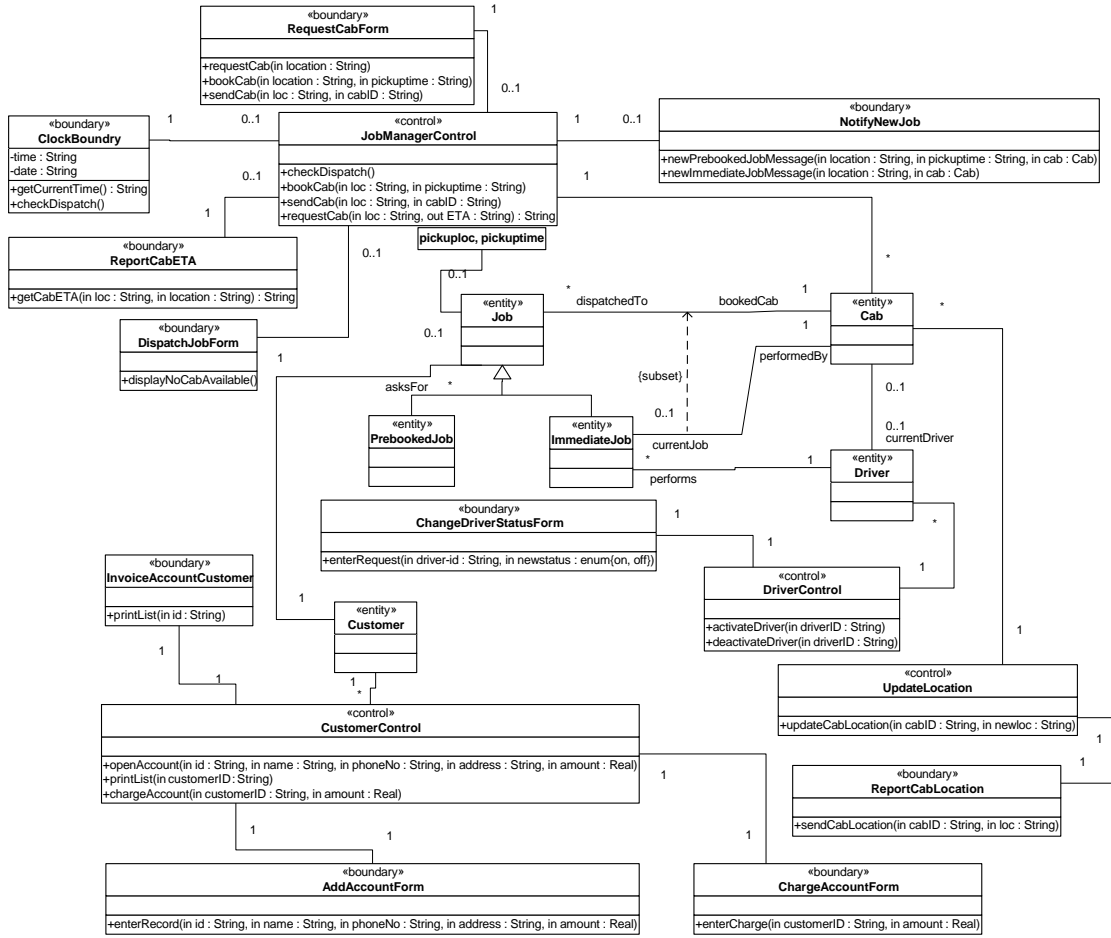
Use Case Name:	PrintInvoiceAccountCustomer
Participating actor:	Initiated by Administrator, communicates with Printer
Entry conditions:	The Administrator asks for the list of all outstanding jobs for an account customer
Flow of events:	<ol style="list-style-type: none"> <li>1. The Administrator inserts the customer's id, and the system checks if the customer exists</li> <li>2. The list of outstanding jobs for the customer is generated.</li> <li>3. The list is spooled to the printer.</li> </ol>
Exit condition:	The list has been successfully printed

### 3. Class Diagram

Two class diagrams are drawn here. The first one is the detailed diagram for the entity classes and the associations among them only. The second one is a complete diagram without details in entity classes.

Each class below is assumed to have a constructor and destructor, as well as get and set methods that can access and set the attribute values of an object. However, they are omitted in the class diagram for clarity.





## 4. Data Dictionary

### 4.1. Entity classes

#### 4.1.1 Cab

A cab is a means of transport from one location to another. A cab is dispatched for a job. A job is created in response to a customer request. A cab is dispatched to carry out a job if an initial confirmation has been received from the customer. A cab can be in one of the four states – *Idle*, *Assigned*, *Busy* or *Off*.

Idle – The cab is available for jobs.

Assigned – The cab has been assigned to a job.

Busy – The cab is performing a job.

Off – The cab is not active.

#### **Attributes:**

**cabID: String**

Cab unique identifier.

**currentloc: String**

A string of characters representing the current location of a Cab.

**cabState: enum{off, idle,  
assigned, busy}**

An enumeration attribute representing possible Cab states (see statechart).

**cabs: Set(Cab)**

Static attribute keeping all instances of Cab

**Methods:****completeJob()**

Description:

Makes a Cab's job completed.

Parameter-List:

Return-List:

**cancelCab(j:PrebookedJob)**

Description:

A Job is cancelled by the Customer before a Cab performs the job.

Parameter-List:

The cancelled job

Return-List:

**selectCabForPrebookedJob(String: pickupTime):Cab**

Description:

Select a cab for a prebooked job.

Parameter-List:

The pickup time of the prebooked job.

Return-List:

A cab that is booked for the job.

**assignCab(j:PrebookedJob)**

Description:

A Cab is assigned to a prebooked job, if the cab does not need to perform other prebooked jobs in 2 hours.

Parameter-List:

The Job to be assigned

Return-List:

**dispatchCab(j:ImmediateJob)**

Description:

A Cab is dispatched to perform a Job, if the cab does not need to perform other prebooked jobs in 2 hours.

Parameter-List:

The Job to be performed

Return-List:

**setLocation(currentloc: String)**

Description:

The Cab's location is reset after the C-GPS sends a new location.

Parameter-List:

currentloc: String representing the new Cab location

Return-List:

**offRoad()**

Description:

The Cab is off-road (e.g., when it is undergoing maintenance)

Parameter-List:

Return-List:

**onRoad()**

Description: The Cab is back on the road and it can be assigned to a job.

Parameter-List:

Return-List:

### **getLocation(): String**

Description:

Parameter-List:

Return-List: The Cab current location

### **getState(): enum{off, idle, standby, busy}**

Description:

Parameter-List:

Return-List: Returns the current state of the Cab

### **listJobs(): Set(Job)**

Description: Returns the list of all the Jobs dispatched to the Cab

Parameter-List:

Return-List: A set of references to Job objects

### **currentJob(): Job**

Description:

Parameter-List:

Return-List: Returns a reference to the Cab's current job

### **driver(): String**

Description:

Parameter-List:

Return-List: Returns the id of the Cab's current driver

### **nextCab(): Cab**

Description: Get the next available cab in the system

Parameter-List:

Return-List: Reference to a Cab object

### **getID(c:Cab): String**

Description: Return the cab's cab ID

Parameter-List: c: the cab

Return-List: the cab's ID

### **getCab(id: String): Cab**

Description: (static) Returns a reference to the Cab having a given id

Parameter-List: id: the Cab's id

Return-List: Reference to a Cab object

### **getFirstCab():Cab**

Description: Get the first available cab  
Parameter-List:  
Return-List: A cab

**isLastCab():Boolean**

Description: (Static) Return if the cab is the last cab in the system to be checked  
Parameter-List:  
Return-List: If the cab is the last cab



### 4.1.2 Customer

Represents a customer of the cab company. Information such as name, customer-id, phone number, and address are recorded.

#### **Attributes:**

<b>name: String</b>	The Customer's name
<b>customer-id: String</b>	A string ID unambiguously identifying the Customer
<b>phoneNumber: String</b>	Customer's phone number
<b>address: String</b>	Customer's address
<b>account: Real</b>	Customer's available credit
<b>customers[]: set(Customer)</b>	Static attribute keeping all instances of the Customer class

#### **Methods:**

##### **updateAccount(amount: Real)**

Description:	Updates the Customer's credit of a given amount (positive or negative)
Parameter-List:	amount: amount to be added to the credit
Return-List:	

##### **getCustomer(id: String):Customer**

Description:	(Static) returns a Customer's object, given the Customer's id
Parameter-List:	id: identifier of the Customer object to search for.
Return-List:	Reference to a Customer object

##### **getNotInvoicedJobs(): Set(Job)**

Description:	
Parameter-List:	
Return-List:	Returns (if exists) the Customer's active Job

### 4.1.3 Driver

A Driver drives a Cab, picks up Customers, and drops them at destination. The Driver charges the Customer account if the Customer chooses to debit his/her account instead of paying cash. A Driver informs the system of the start and end of a shift.

#### **Attributes:**

**driverID: String** String unambiguously identifying a Driver  
**driverMode: enum{on, off}** State of the driver: indicates if the Driver is active  
**drivers[]: Set(Driver)** Static attribute keeping all instances of the class Driver

#### **Methods:**

**changeMode()**  
 Description: Flips the Driver mode.  
 Parameter-List:  
 Return-List:

#### **getDriver(id: String): Driver**

Description: (Static) returns a Driver object, given its id  
 Parameter-List: id: ID of the Driver to be retrieved  
 Return-List: Reference to a Driver object

#### **performedJobs(): Set(Job)**

Description: Returns all the Jobs performed by the Driver  
 Parameter-List:  
 Return-List: A set of references to Job objects

#### 4.1.4 Job

A Job is created in response to a customer enquiry for cabs.

##### **Attributes:**

<b>pickuploc: String</b>	A String representing pickup location of a Customer.
<b>deliveryloc: String</b>	A String representing delivery location of the Customer.
<b>phoneNumber: String</b>	A String representing a telephone number of the Customer.
<b>pickupTime: String</b>	A String representing Customer's pick up time
<b>ETA: String</b>	Estimated time of arrival to Customer's location
<b>financialStatus: enum{ notinvoiced, invoiced, paid}</b>	Status of the Jobs' invoice
<b>jobs[]: Set(Job)</b>	Static attribute keeping all instances of the class Job

##### **Methods:**

##### **getBookedCab():Cab**

Description:	Get the booked cab for the job
Parameter-List:	
Return-List:	The cab booked for the job

##### **getPickuploc(): String**

Description:	Get the pickup location of a job
Parameter-List:	
Return-List:	The pickup location of a job

##### **getPickuptime(): String**

Description:	Get the pickup time of a job
Parameter-List:	
Return-List:	The pickup time of a job

#### 4.1.5 ImmediateJob

This is a specialization of class Job. An instance of class ImmediateJob is created when a customer asks for a Cab from location A to location B.

##### **Attributes:**

<b>jobState:enum {dispatched, complete}</b>	A cab has been sent to pickup the customer The customer has been delivered to the destination
---	--

##### **Methods:**

##### **ImmediateJob(cabID:String, loc:String)**

Description:	Constructor
Parameter-List:	cabID: The id of the cab that has to perform the Job loc: the pickup location
Return-List:	

**completeJob()**

Description:

Complete an immediate Job

Parameter-List:

Return-List:

### 4.1.6 PrebookedJob

This is another specialization of the class Job. An instance of PrebookedJob is created if a Customer wants to pre-book a cab. A Job is created and recorded.

### Attributes:

**cancelTime: String**

<b>jobState:enum {pending,</b>	A prebooked job is created
<b>cancelled,</b>	A prebooked job is cancelled
<b>confirmed}</b>	A prebooked job is confirmed and to be performed

## **Methods:**

### **PrebookedJob(cabID:String, loc:String, time:String)**

Description:	Constructor
--------------	-------------

Parameter-List: cabID: The id of the cab that has to perform the Job  
loc: the pickup location  
time: the pickup time

Return-List:

**verifyPreJob(): Boolean**

Description:	Verifies if the prebooked Job should be confirmed, i.e. between 15 to 20 minutes before pickup time
--------------	---

Parameter-List:

Return-List: True if the Job should be confirmed, false otherwise

### **getFirstPrebookedJob(): PrebookedJob**

Description:	(static) Returns the first instance of PrebookedJob
--------------	---

Parameter-List:

Return-List:	The first instance of PrebookedJob (if it exists)
--------------	---

### getNextPrebookedJob(): PrebookedJob

<b>Description:</b>	(static) Returns the next instance of PrebookedJob
---------------------	--

### Parameter-List:

**Return-List:** The next instance of PrebookedJob

## cancel()

Description: Cancel the pre-booked job. (it must be done at least 24 hours before the pickup time)

Parameter-List:

Return-List:

## confirmJob()

Description: To confirm the prebooked job

### Parameter-List:

Return-List:

**isLastPrebookedJob():Boolean**

Description: (static) To confirm if the prebooked job is the last prebooked job to be checked

Parameter-List:

Return-List: Return if the prebooked job is the last one

## 4.2. Control Classes

### 4.2.1 CustomerControl

Handles basic operations involving a Customer: opening a new account, charging an amount to a Customer's account, refilling an account, and printing the list of invoices to be sent to the Customer.

#### **Attributes:**

#### **Methods:**

**openAccount(id: String, name: String, phoneNo: String, address: String, amount: Real)**

Description: Creates a new Customer's account (i.e., a new instance of Customer)

Parameter-List: id: A string ID unambiguously identifying the Customer  
 name: The Customer's name  
 phoneNo: Customer's phone number  
 address: Customer's address  
 amount: Customer's initial credit

Return-List:

**printList(customerID: String)**

Description: Prints the list of all invoices to be sent to a Customer

Parameter-List: id: identifier of the Customer for which the list is to be printed

Return-List:

**chargeAccount(customerID: String, amount: Real)**

Description: Charges a Customer's account of a given amount

Parameter-List: customerID: ID of the Customer the account is to be charged  
 amount: amount to charge

Return-List:

#### 4.2.2 DriverControl

A Cab is allocated or booked in response to Customers telephone call. A Driver is activated when starting a new shift and deactivated when the shift is finished. In addition, the Expected Time of Arrival (ETA) of Cab is given to the Customer.

##### **Attributes:**

##### **Methods:**

##### **activateDriver(driverID: String)**

Description: A Driver starts a new shift  
Parameter-List: DriverID: ID of the Driver to be activated  
Return-List:

##### **deactivateDriver(driverID: String)**

Description: A Driver ends his/her shift for the day.  
Parameter-List: DriverID: ID of the Driver to be deactivated  
Return-List:



### 4.2.3 JobManagerControl

The JobManagerControl class manages Jobs (creation, booking) and is responsible for dispatching cabs.

#### **Attributes:**

#### **Methods:**

##### **checkDispatch()**

Description: Dispatch pre-booked Jobs to be dispatched

Parameter-List:

Return-List:

##### **bookCab(loc: String, pickuptime: String):String**

Description: A Customer pre-books a cab

Parameter-List: loc: pickup location

pickuptime: pickup time

Return-List: The ID of the cab that has been assigned to a job

##### **sendCab(loc: String, cabID: String)**

Description: A Cab is sent to service an immediate Job

Parameter-List: loc: the location to be reached

cabID: the id of the Cab that will perform the Job

Return-List:

##### **requestCab(loc: String, ETA: String):String**

Description: Asks for the available cab with the smallest ETA to be immediately dispatched to pickup a Customer at a given location (once a cab is identified, sendCab can be used)

Parameter-List: loc: the location to be reached

ETA: the ETA of the cab found (output parameter)

Return-List: The cab's ID

### 4.2.4 UpdateLocationControl

This is used by satellite to update the current location of Cab.

#### **Attributes:**

#### **Methods:**

##### **UpdateCabLocation(cabID: String, newloc: String)**

Description: Updates the Cab location stored in the system

Parameter-List: cabID: unique identifier of the Cab to be updated

newloc: the new location of the Cab

Return-List:

### 4.3. Boundary Classes

#### 4.3.1 AddAccountForm

The administrator uses this object to add new customer account

**Attributes:**

**Methods:**

**enterRecord(id: String, name: String, phoneNo: String, address: String, amount: Real)**

Description: Creates a new Customer's account

Parameter-List: id: A string ID unambiguously identifying the Customer  
 name: The Customer's name  
 phoneNo: Customer's phone number  
 address: Customer's address  
 amount: Customer's initial credit

Return-List:

#### 4.3.2 ChargeAccountForm

The driver uses this to charge Customer's account.

**Attributes:**

**Methods:**

**enterCharge(customerID: String, amount: Real)**

Description: Used to charge a Job's amount to a Customer

Parameter-List: customerID: ID of the Customer the account is to be charged  
 amount: amount to charge

Return-List:

### 4.3.3 DispatchJobForm

Used by JobManager for immediate Cab dispatching.

**Attributes:**

**Methods:**

**displayNoCabAvailable()**

Description: A return message is sent to the user to indicate that no Cab is currently available

Parameter-List:

Return-List:

### 4.3.4 InvoiceAccountCustomer

The administrator uses this to request for account customers list, spool it to the printer spooler and eventually print the list.

**Attributes:**

**Methods:**

**printList(id: String)**

Description: Asks the system to print the list of invoices to be sent to a Customer

Parameter-List: id: the Customer's identifier

Return-List:

#### 4.3.5 ReportCabLocation

This is used by the Calendar and Clock System to communicate the current location of the Cab

##### **Attributes:**

##### **Methods:**

**sendCabLocation(cabID: String, loc: String)**

Description: CGPS sends the current Cab location to the system.

Parameter-List: cabID: unique id of the Cab to be updated

loc: new Cab location

Return-List:

#### 4.3.6 ChangeDriverStatusForm

The Dispatcher uses this to communicate with the System when a Driver indicates a change of status

##### **Attributes:**

##### **Methods:**

**enterRequest(driverID: String, newstatus: enum{on, off})**

Description: Asks the system to change the Driver's status

Parameter-List: driverID: ID of the Driver the status is to be changed

newstatus: new Driver's status

Return-List:

### 4.3.7 RequestCabForm

This is the request Cab interface.

**Attributes:**

**Methods:**

**requestCab(location: String)**

Description: Dispatcher requests for a Cab to be allocated to a new Job.  
 Parameter-List: location: Location to reach for picking-up the Customer  
 Return-List:

**bookCab(location: String, pickuptime: String)**

Description: Dispatcher books a Cab for a pre-booked Job  
 Parameter-List: location: Location to reach for picking up the Customer  
 pickuptime: time of the rendezvous  
 Return-List:

**sendCab(loc: String, cabId: String)**

Description: A Cab is sent to service an immediate Job.  
 Parameter-List: loc: the pickup location  
 cabID: the cab's id  
 Return-List:

### 4.3.8 NotifyNewJob

Interface used by class JobManagerControl to send new job messages to onboard computers placed on Cabs

#### **Attributes:**

#### **Methods:**

##### **newPrebookedJobMessage(location: String, pickuptime: String, cab: Cab)**

Description: Sends a message to notify a new pre-booked Job  
 Parameter-List: location: Location to reach for picking up the Customer  
 pickuptime: time of the rendezvous  
 Return-List:

##### **newImmediateJobMessage(location: String, cab: Cab)**

Description: Sends a message to notify a new immediate Job  
 Parameter-List: location: Location to reach for picking up the Customer  
 Return-List:

### 4.3.9 ReportCabETA

Communicate with the GIS to get the ETA of a Cab to a give location

#### **Attributes:**

#### **Methods:**

##### **getCabETA(loc: String, location: String): String**

Description: GIS sends the ETA of a Cab to a given location.  
 Parameter-List: loc: a Cab's location  
 location: the pickup location  
 Return-List: The ETA of a Cab's location to the given location

### 4.3.10 ClockBoundry

This is used by the Calendar and Clock System to communicate the current time and trigger servicing prebooked job.

#### **Attributes:**

time:String  
 date:String

**Methods:****getCurrentTime(): String**

Description: Calendar and Clock System gives the current time.

Parameter-List:

Return-List: The current time.

**checkDispatch()**

Description: Activates a pre-booked Job so that a Cab can perform the Job.

Parameter-List:

Return-List:

Special requirement: checkDispatch() should be triggered every minute.

#### 4.4. Associations between entity classes

Source: **Cab**  
 Target: **Driver**  
 Description: Association between a Cab and its current Driver. A Cab in Off state has no Driver , and a Driver is associated with a Cab only during his/her shift

Role source:  
 Role target: currentDriver  
 Cardinality source: 0..1  
 Cardinality target: 0..1

Source: **Driver**  
 Target: **ImmedicateJob**  
 Description: All the jobs performed by a driver.  
 Role source:  
 Role target: performs  
 Cardinality source: 1  
 Cardinality target: \*

Source: **Cab**  
 Target: **Job**  
 Description: A Cab can be booked for many jobs; while a job has one cab associated  
 Role source: bookedCab  
 Role target: dispatchedTo  
 Cardinality source: 1  
 Cardinality target: \*

Source: **Cab**  
 Target: **ImmediateJob**  
 Description: An immediate job is performed by one cab; while a cab can have zero or one immediate job at a time.  
 Role source: performedBy  
 Role target: currentJob  
 Cardinality source: 1  
 Cardinality target: 0..1



Source:	<b>Customer</b>
Target:	<b>Job</b>
Description:	A Customer may ask for a Cab's Job.
Role source:	asksFor
Role target:	
Cardinality source:	1
Cardinality target:	*

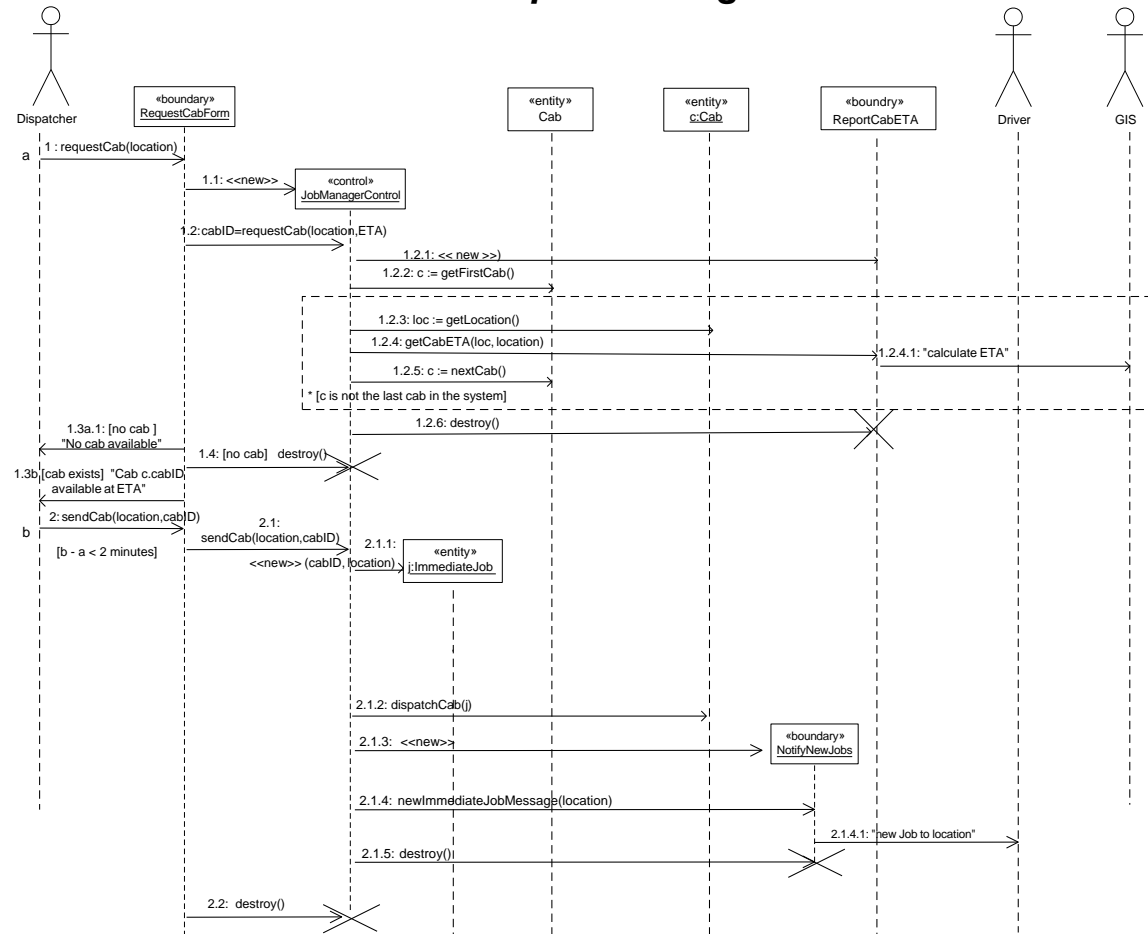
## 5. Sequence diagrams

### 5.1. Abbreviations

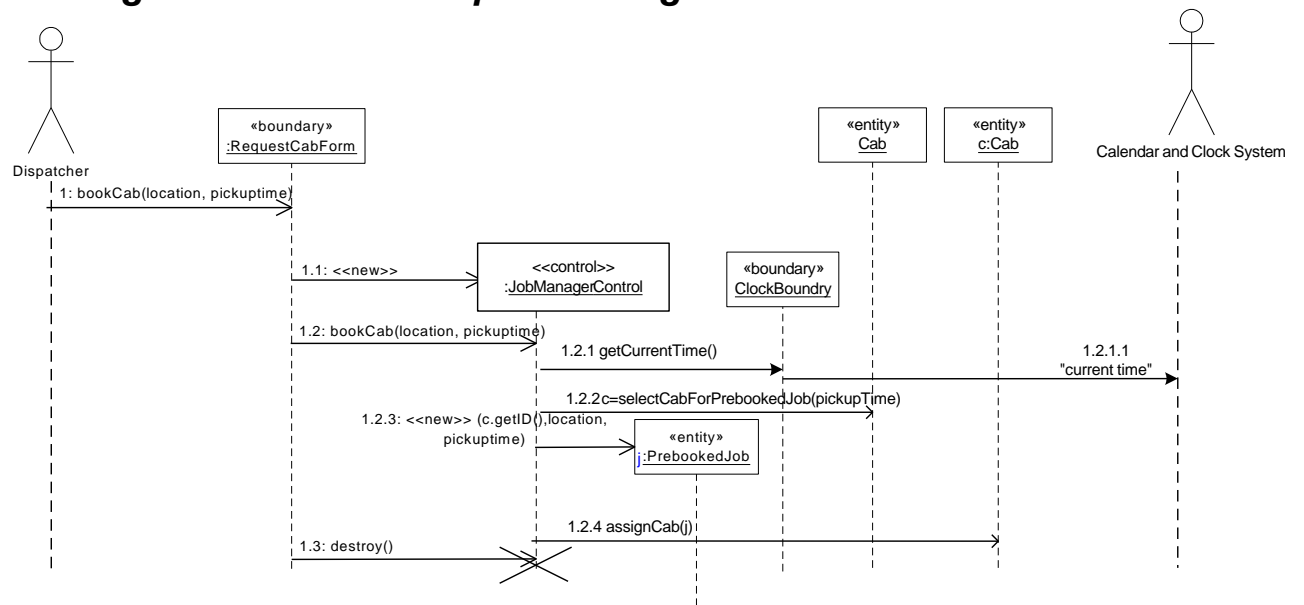
It's going to be easier to read the models if we always use the same letter throughout for a given model type.

Variable	Type
c	Cab
j	Job

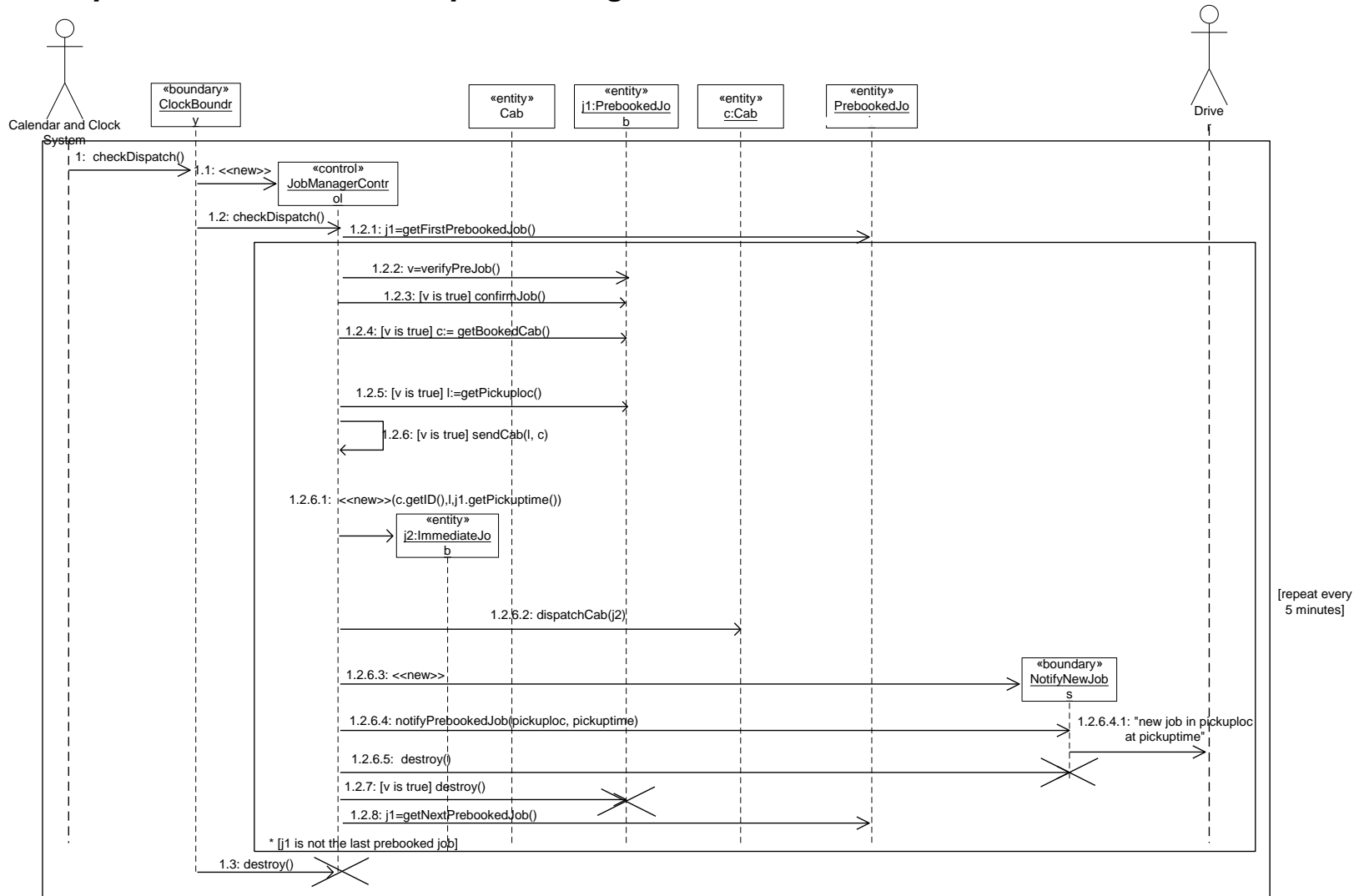
### 5.2. HandleImmediateJob Sequence Diagram



### 5.3. LogPrebookedJob Sequence Diagram

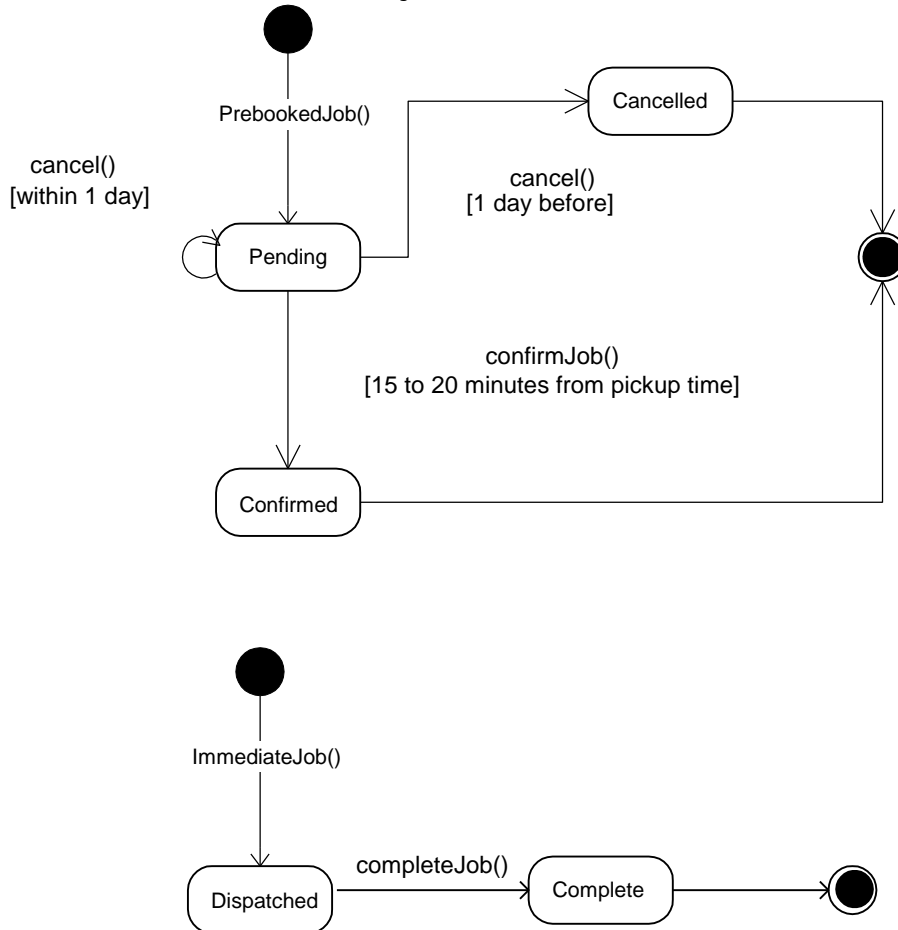


## 5.4. DispatchPrebookedJob Sequence Diagram



## 6. State charts

### 6.1. State chart for an object of class Job

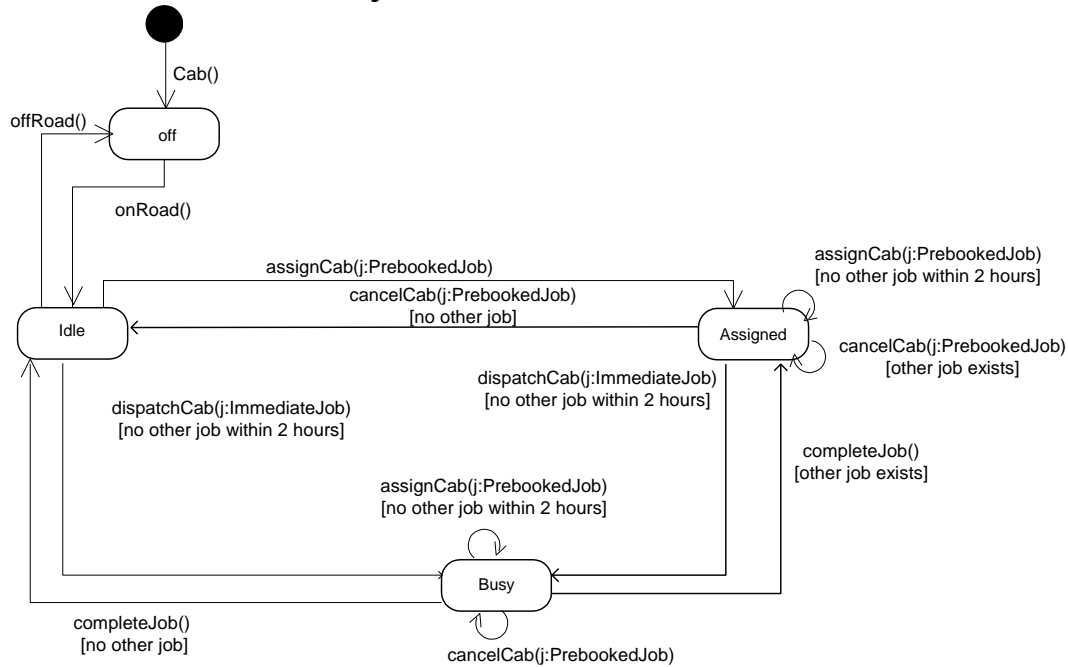


#### Short description

The constructor (`PrebookedJob()`) is called to schedule a new Job. Once created, the Job is in *Pending* state. A `confirmJob()` operation confirms the Job. A `cancel()` operation cancels the Job, provided that it is done soon enough.

The constructor (`ImmediateJob()`) is called to create a new Job. Once created, the Job is in *Dispatched* state. A `CompleteJob()` operation finishes the Job.

## 6.2. Statechart for an object of class Cab



### Short description

A new Cab object, instantiated by constructor `Cab()`, is by default in the *Off* state. When the driver reports an new shift, the `onRoad()` method make the Cab *Idle* and waiting for a Job. On the other hand, The `offRoad()` method switches Off the Cab when the driver reports the end of the shift.

From the *Idle* state, a Cab is assigned to a prebooked job via `assignCab(j:PrebookedJob)`, going to the *Assigned* state. More than 1 prebooked jobs can be assigned to a Cab. From *Assigned* state, when a job is cancelled via `cancelCab(j:PrebookedJob)`, the Cab goes back to the *Idle* state, if there is no other job assigned to it.

A Cab can also be dispatched to an immediate job, going to the *Busy* state. When a job is completed via `completeJob()`, the Cab goes back to the *Idle* state, if there is no other job assigned, otherwise *Assigned* state.