Chapter 1

Algorithm Design

This section is intended to better explain some functions and algorithms that can be ambiguous in the previous presentation of our system.

Algoritmi speciali usati nel nostro sistema, uno per uno descritti con pseudocodice o flow chart

algoritmi che potremmo mettere:

- quello per selezionare il messaggio e gestirlo (IO Manager)

  1)procedure selectMessage()

- taxi allocator (selezione delle ride non gestite e gestione)
- queues manager (check positions e ordinamento delle code)
- modify priority –¿ recall della funzione di ordinamento del queue manager
- credentials checker (check email, password)

Algorithm 1 Descrizione Algoritmo

```
1: procedure EXAMPLE(par1, par2)                    ▷ example algorithm
2:     par1 ← 1
3:     if i ≥ maxval then
4:         i ← 0
5:     else
6:         if i + k ≤ maxval then
7:             i ← i + k
8:         end if
9:     end if
10: end procedure
```

1.1 IO Manager Algorithms

Here we describe some functions and algorithms relatives to the IO Manager component.

1.1.1 Select Message

The select message procedure have to select the right message handler for an incoming/outgoing message. It's described by the following pseudocode:

Algorithm 2 Select Message Procedure

1: **procedure** SELECTMESSAGE($rawMessage$) ▷ rawMessage is an xml (or other markup language like json) message
2: $messageHandler \leftarrow messagesTypeMap.getMessageHandler($
3: $rawMessage.type)$
4: $returnmessageHandler.handleMessage(rawMessage)$
5: **end procedure**

Note on handleMessage(): every single message class that is present on the IO Manager component, can handle incoming messages or outgoing messages and the implementation of handleMessage() for each of this sub-components is different depending on the message that has to be handled.

1.2 Requests and Reservations Manager Algorithms

Here we describe some functions and algorithms relatives to the Requests and Reservations Manager component.

1.2.1 Check Rides To Allocate

The checkRidesToAllocate procedure is invoked by a scheduler every 2 minutes. This procedure has to check if there are rides or reservations that has to be handled and to put them into the ride-handling stack.

Algorithm 3 Check Rides to Allocate

1: **procedure** CHECKRIDESTOALLOCATE
2: $tempList \leftarrow DatabaseManager.getReady()$ ▷ get all the rides that need to be handled
3: **for all** $element$ in $tempList$ **do**
4: $allocateRide(element)$ ▷ handle the single ride
5: **end for**
6: **end procedure**

1.2.2 Allocate Ride

Allocate Ride starts a new Thread to perform the handling in parallel with the others operations.

Algorithm 4 Allocate Ride

```
1:  procedure ALLOCATERIDE(ride)
2:      taxiID ← 0                                              ▷ initialization
3:      repeat
4:          taxiID ← ZonesManager.getFirstAvailableTaxiDriver()
5:          msgTD ← createMessage("TaxiDriverNotif", taxiID)
6:          response ← IOManager.selectMessage(msgTD)
7:          ZonesManager.changePriority(0, taxiID)
8:      until response == decline
9:      DatabaseManager.setAvailability(taxiID, false)
10:     msgC ← createMessage("CustomerNotif", ride.CustomerID)
11:     IOManager.selectMessage(msgC)
12: end procedure
```

1.3 Zones Manager Algorithms

Here we describe some functions and algorithms relatives to the Zones Manager component.

1.3.1 Check Positions

This procedure is invoked by a scheduler every 2 minutes and updates the zones of every Taxi Driver.

Algorithm 5 Check Positions

```
1:  procedure CHECKPOSITIONS
2:      availableDrivers ← DatabaseManager.getAvailableTDs()
3:      for all dirver in availableDrivers do
4:          driverZone ← ZoneCalculator.calculateZone(
5:          driver.position)
6:          driver.setZone(driverZone)
7:      end for
8:      orderQueues(availableDrivers)
9:  end procedure
```

1.3.2 Order Queues

This procedure has to order the queues in function of the priority of the Taxi Drivers.

Algorithm 6 Order Queues

```
1: procedure orderQueues(drivers)
2:     QueuesManager.resetQueues()
3:     setCurrentDrivers(drivers)
4:     for all driver in drivers do
5:         zoneQueue ← QueuesManager.getQueue(driver.zone)
6:         zoneQueue.orderedInsert(driver)  ▷ insert the driver ordering by the priority
7:     end for
8: end procedure
```

1.3.3 Modify Priority

This procedure has to modify the priorities of the current queued Taxi Drivers and reinsert the driver in his queue with ordered

Algorithm 7 Modify Priority

```
1: procedure modifyPriority(new, taxiID)
2:     driver ← getFromCurrentDrivers(taxiID)
3:     driverQueue ← QueueManager.getQueue(driver.zone)
4:     driverQueue.remove(taxiID)              ▷ remove from the current zone
5:     driver.setPriority(new)
6:     driverQueue.orderedInsert(driver)                ▷ reinsert in order
7: end procedure
```

1.4 Profile Manager Algorithms

Here we describe some functions and algorithms relatives to the Profile Manager component.

1.4.1 Check Email

This procedure is in charge to tell if the given email is a valid one.

Algorithm 8 Check Email

```
1: procedure checkEmail(emaik)
2:     regex ← "[a − zA − Z1 − 9] + @[a − zA − Z]?.[a − zA − Z]2, 3"
3:     if regex.match(email) then
4:         return true
5:     end if
6:     return false
7: end procedure
```

1.4.2 Check Password

This procedure is in charge to tell if the given password is a valid one.

Algorithm 9 Check Password

1: **procedure** CHECKPASSWORD($password$)
2: $length \leftarrow password.length()$
3: $numbers \leftarrow containsDigits(password)$
4: $letters \leftarrow containsCharacters(password)$
5: return $((length > 8)$ and $numbers$ and $letters)$
6: **end procedure**

1.4.3 Check Username

This procedure is in charge to tell if the given username is a valid one.

Algorithm 10 Check Username

1: **procedure** CHECKUSERNAME($username$)
2: return $(username.length() > 8)$
3: **end procedure**