Software Architectures 2021-22

Project 1

Theme: Health Centre
Authors: Mário Silva nºmec 93430, Daniel Gomes nºmec 93015
Date: 15/04/2022

Index

1 Introduction

This project was developed in the context of the Software Architectures course for the year 2021/22. The goal of the work consists of the simulation of the management process in a small Health Centre. It is based on an evaluation process carried out by nurses, where each patient is assigned a degree of severity (DoS) before a medical appointment. A patient arrives at the health centre and proceeds to the Entrance Hall, where he waits for his turn. Upon a call from the Call Centre, a patient is evaluated regarding their DoS, and then he proceeds to the Waiting Hall. Upon a call from the Call Centre, the patient proceeds to the Medical Hall and waits for his turn. Finally, the patient pays for the respective medical appointment.

2 Processes and Entities Involved

The implementation carried out consisted of 2 processes: a Control Centre and a Health Centre. Following the Client-Server pattern, the Control Centre and Health Centre have the client and server roles, respectively. Thus, the Control Centre is responsible for controlling the state of the simulation, for instance, when it should start, stop, resume, terminate or switch between Manual and Automatic mode. On the other hand, the Health Centre manages the logic to supervise the concurrency between patients, and also the access to the Shared Region with the other entities. These remaining entities are the Nurses, Doctors, the Call Centre and the Cashier. Further, in this document, the monitors created and the interaction of all entities with them will be described and explained.

3 Communication

To perform the communication between the processes, TCP sockets have been used. In the figure below, it is shown the initial workflow implemented between the Health Centre and Control Centre.

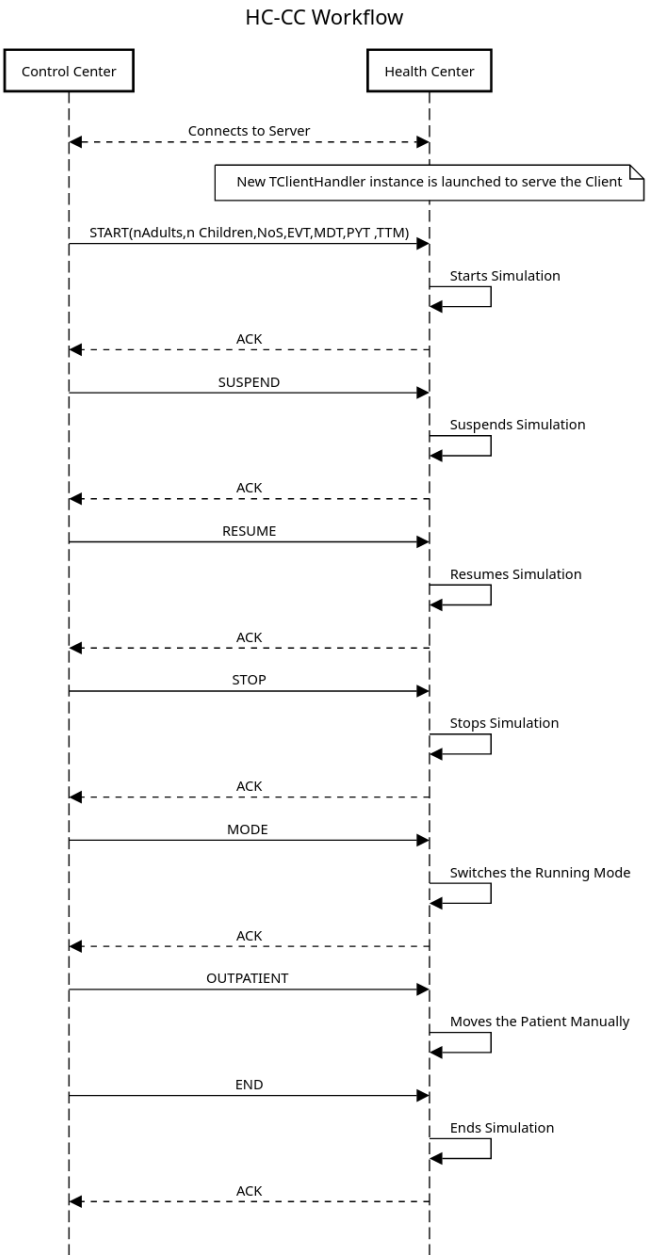

Figure 1: Interaction between the Call Centre and Health Centre

As it may be seen, whenever a client wants to connect to the server, a new thread of ***ClientHandler*** is launched, to attend the respective client. From now on, the client may exchange messages to control the state of the simulation. Each of these messages is sent using an object of the class ***Message***, which accepts the following topics:

- START, to start the simulation.
- SUSPEND, to suspend the simulation: Patients, Call Center and Cashier stop their activity.

- STOP, to stop the simulation: simulation evolves to its initial state and all Patients die.
- RESUME, to resume the suspended simulation: Patients, Call Centre and Cashier resume their normal activity,
- END, both processes die.
- MODE, alternate the operating mode between manual and automatic,
- OUTPATIENT, allow the movement of one patient while in manual mode.

Consequently, the respective ***ClientHandler*** responsible, according to the topic of the message, will perform different kinds of actions. For example, whenever the START topic is sent, the ***ClientHandler*** will instantiate all Monitors required for the simulation, as well as the correct number of threads for each of the entities involved.

4 Control Centre

The Control Centre, as mentioned before, control the state of the simulation. To make it more visually appealing, a Graphical User Interface(GUI) has been created, using *Java Swing*. This interface may dynamically change according to the events involved in the simulation. For example, whenever a new simulation starts, the *Start* button present on the interface will be disabled, avoiding that any user starts a new one, without stopping a running simulation. Besides this, the input fields for the parameters of the simulation, such as the Number of Patients and Seats, will be disabled too. The following figure shows what the Graphical Interface of the Control Centre looks like.

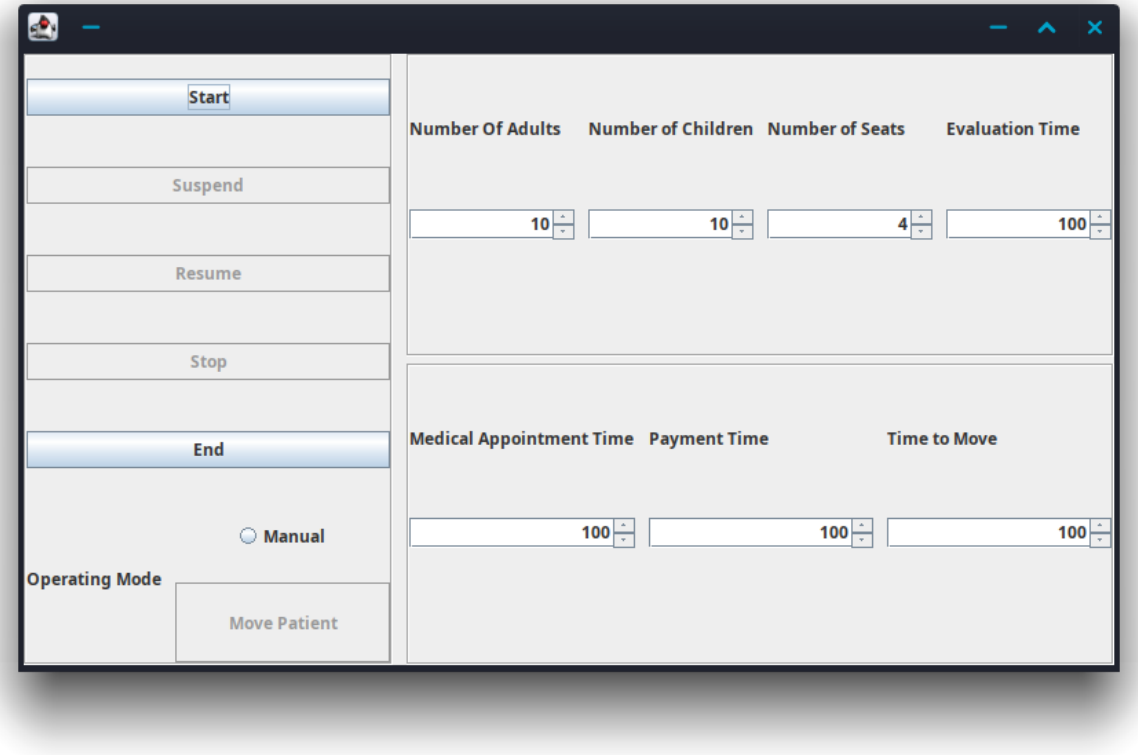

Figure 2: Control Centre GUI

5 Health Centre

Similarly to the Control Centre, the Health Centre also has a Graphical Interface on *Java Swing*, which is popped up on the user's screen, whenever a new simulation starts. The goal of this interface is to visualize the behaviour of each Patient thread, from the first stage of the simulation, the *Entrance Hall*, to the last stage, the *Payment* Hall. The interface appearance can be seen in the figure below.

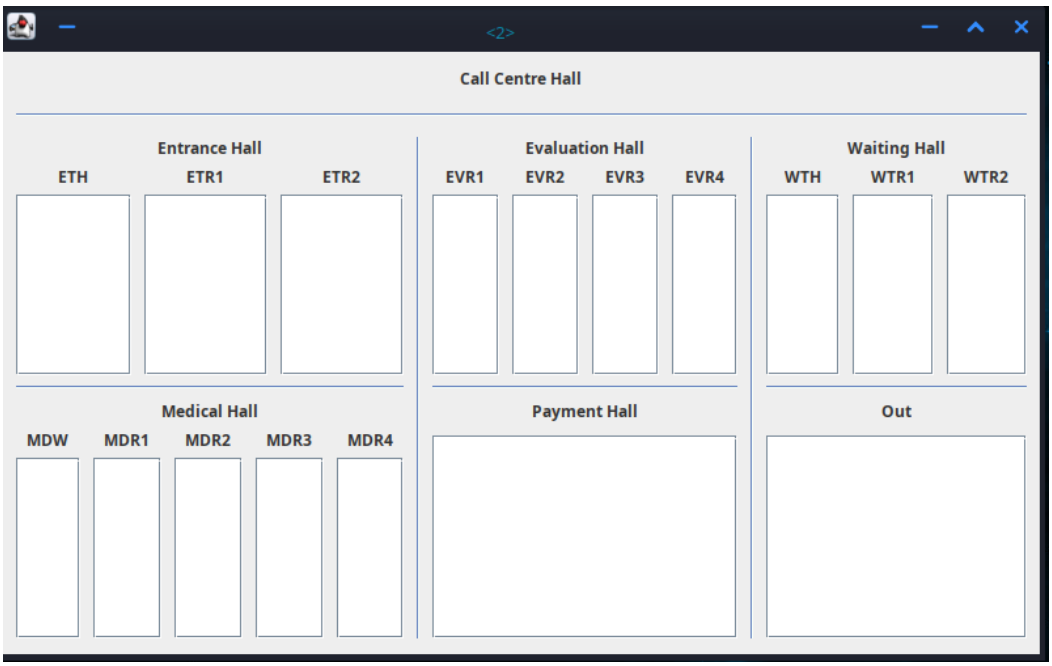

Figure 3: Health Centre GUI

Regarding, the *Back-end* side of the Health Centre, which, as mentioned in previous subsections, contains the logic of the life-cycle of the Hospital, 7 monitors have been developed:

- Controller
- Call Centre Hall
- Entrance Hall
- Evaluation Hall
- Waiting Hall
- Medical Hall
- Payment Hall

5.1 Monitors

On this section, it will be explained the details of the implementation of the Monitors that have been mentioned on the previous subsection: its purpose, the entities involved and the actions taken from each of these, and consequently the workflow taken. For each entity that needs to interact with a certain Monitor, an interface has been created with the following nomenclature: I*MonitorName_EntityName*, that would be implemented by the respective Monitor.

- Controller
  - Entities Involved: All Entities.

  - Shared Region: This monitor enables a deeper control of the Health Centre flow, and its logging. It achieves this by providing the lock used by the Call Centre (Entity) and on the Call Centre Hall. It also has access to the interface methods, therefore responsible for its updates.
  - Workflow: Entities use this monitor to update their current positions in the interface and the log file as well. It also contains a method responsible for allowing the suspension of the simulation, by alternating a boolean and all entities check for suspension before and after sleeping, entering rooms/halls and being signalled from other conditions. This monitor allows changing the operation mode as well, during the manual mode, it is possible to activate or deactivate the call centre allowing one patient to move.

- Call Centre Hall
  - Entities Involved: **Patient** and the **Call Centre**.
  - Shared Region: The shared region of the Call Centre Hall contains a *HashMap*, *hallPatients*, to store the number of patients present in each Hall with a *String* with the identification of each Hall, as key. This data structure will allow the Call Centre Hall to have control over the number of patients in each Hall/Room.
  - Workflow: Every time a patient notifies the Call Centre Hall of its entrance or exit, the *HashMap* will be updated with the new number of patient, on the respective hall, and consequently, the Call Centre will start its duty. Thus, regarding the maximum number of seats in each Hall/Room and current number of patients in each of those, the Call Centre will check how many patients it may call to proceed to next Monitor of its lifecyle, and notify them.
- Entrance Hall
  - Entities Involved: **Patient** and the **Call Centre**.
  - Shared Region: The Shared Region of this monitor, will contain two *PriorityQueues*, one for each room (*ETR1*,*ETR2*), as well as two counters of the number of patients present on each PriorityQueues, to control if the rooms are full or not, and the ETN number for both Children and Adults (*ETN1* and *ETN2*), to be assigned to each patient.
  - Workflow: Initially, the patients are assigned with an ETN(*Entrance Number* are assigned to the respective room, ETN1 for Children and ETN2 for Adults, if the rooms are not full. Otherwise, they will remain blocked waiting to enter a room. Once a patient is in a room, it will notify the Call Centre Hall with its arrival, and stay blocked in the room. The Call Centre, will do its role, and verify if there are empty seats on the Evaluation Hall. Consequently, the patient will be notified if that's the case, emptying its place on the room and proceeding to next Monitor.
- Evaluation Hall
  - Entities Involved: **Patient** and the **Nurse**.
  - Shared Region: The Shared Region of the Evaluation Hall will contain an *PriorityQueue* of Patients representing each room(EVR1, EVR2, EVR3 and EVR4), since every room can only have 1 patient at a time.
  - Workflow: Once a new patient arrives at the evaluation hall, an index of the *Array* mentioned above will be occupied by the patient. In other words, an empty room will be assigned to the patient. As there is a nurse for each of the EVRs, the nurse responsible to the room where the patient has been assigned will be signaled, and patient will stay blocked on the room. Consequently, the respective

nurse will evaluate the patient, taking a random time regarding the EVT parameter, and attaches a random DoS to the patient. Finally, the patient leaves the respective EVR and notifies the Call Centre Hall of its exit from the Evaluation Hall.

- Waiting Hall
  - Entities Involved: **Patient** and the **Call Centre**.
  - Shared Region: The Shared Region of the Waiting Hall contains 2 *PriorityQueues* (*WTR1*,*WTR2*), one for the Children room and other for Adults room. Besides this, two counters to represent the amount of patients in each room are used, so that it is possible to control whether the rooms are Full or Not. Finally, the WTN1 and WTN2, are also assigned to Children and Adults, respectively, to keep track of the arrival order.
  - Workflow: The workflow on Waiting Hall is identical to the Entrance Hall. The differences consist in the arrival number assigned, the WTN, and in the way that the patients are retrieved from the Rooms to next Monitor: Firstly, it is taken into account the DoS, as patients with Higher DoS have higher priority. The next criteria to leave the room is the WTN, which, as mentioned above, represents the arrival order of the Patients.
- Medical Hall
  - Entities Involved: **Patient**, **Call Centre** and the **Doctor**.
  - Shared Region: The Shared Region of the Medical Hall contains 2 *Arrays*, *MDRC* and*MDRA*), in which the first represents two rooms for Children (MDR1 and MDR2), and the second one the remaining two rooms for Adults (MDR3 and MDR4). As the Call Centre Hall can supervise how many people are in each Hall/Room, these two variables are enough to represent all rooms. Once again, two counters are also used to keep track if the rooms are full or not.
  - Workflow: Initially, the arriving patients will stay blocked waiting for the Call Centre to call them to go the respective MDRs. When this event occurs, the patients will notify the Call Centre Hall that they have left the *MDWA*, if the patient room was destinated for adults, or *MDWC* otherwise. Then, acordding to the patient type and the available seats, the patient will be assigned to a room: an index of one of the arrays mentioned above. As there is a doctor for each of the rooms, the doctor responsible for the room where the patient is, will be signaled, allowing the doctor to do its role. The medical appointment can now start, taking a random time considering a range defined on the EVT parameter. After the evaluation, the patient will leave the room where has been, and notify the Call Centre Hall that he has left the MDRA or MRDC, according the patient type.
- Payment Hall
  - Entities Involved: **Patient** and the **Cashier**.
  - Shared Region: The Shared Region of the Payment Hall contains a Payment Queue, which is an Array with a size corresponding to the number of seats parameter. As this data structure had a fixed number of elements it can store it was decided to use that value, which means that the if there are more patients waiting to do the payment than the size of the queue, those would be blocked waiting to enter the queue and finally pay to the Cashier. Besides this queue, a counter of the number of occurring payments was also used, which helps control that only one payment can occur at a time, as well as the PYN counter, to keep track of the arrival order of the patients. Once again the counters for controlling the size of the queues were also present.

– Workflow: In this final monitor, initially the patients arrive and are assigned with a PYN, according to their arrival order. If the paymentQueue is not full, these will be put on the queue and signal the Cashier to receive a payment, and stay blocked on the queue. The cashier will then check, the highest priority patient on the queue (the one with the lowest PYN) and take a random time considering the range defined on the PYT parameter. After this time, the patient will leave the queue and end its lifecycle.

5.2 Interaction Diagram

Regarding the monitors mentioned in the previous section, on the figures below it is possible to see high level interaction diagrams of the Monitors. As there were a considerable amount of monitors, the diagram was divided in two sub diagrams, to facilitate the understanding and visualization of the process. Besides this, it should be mentioned that the *Control* Monitor is not present in these diagrams since it does not represent a Hall and the Monitor's role is different, as mentioned previously.

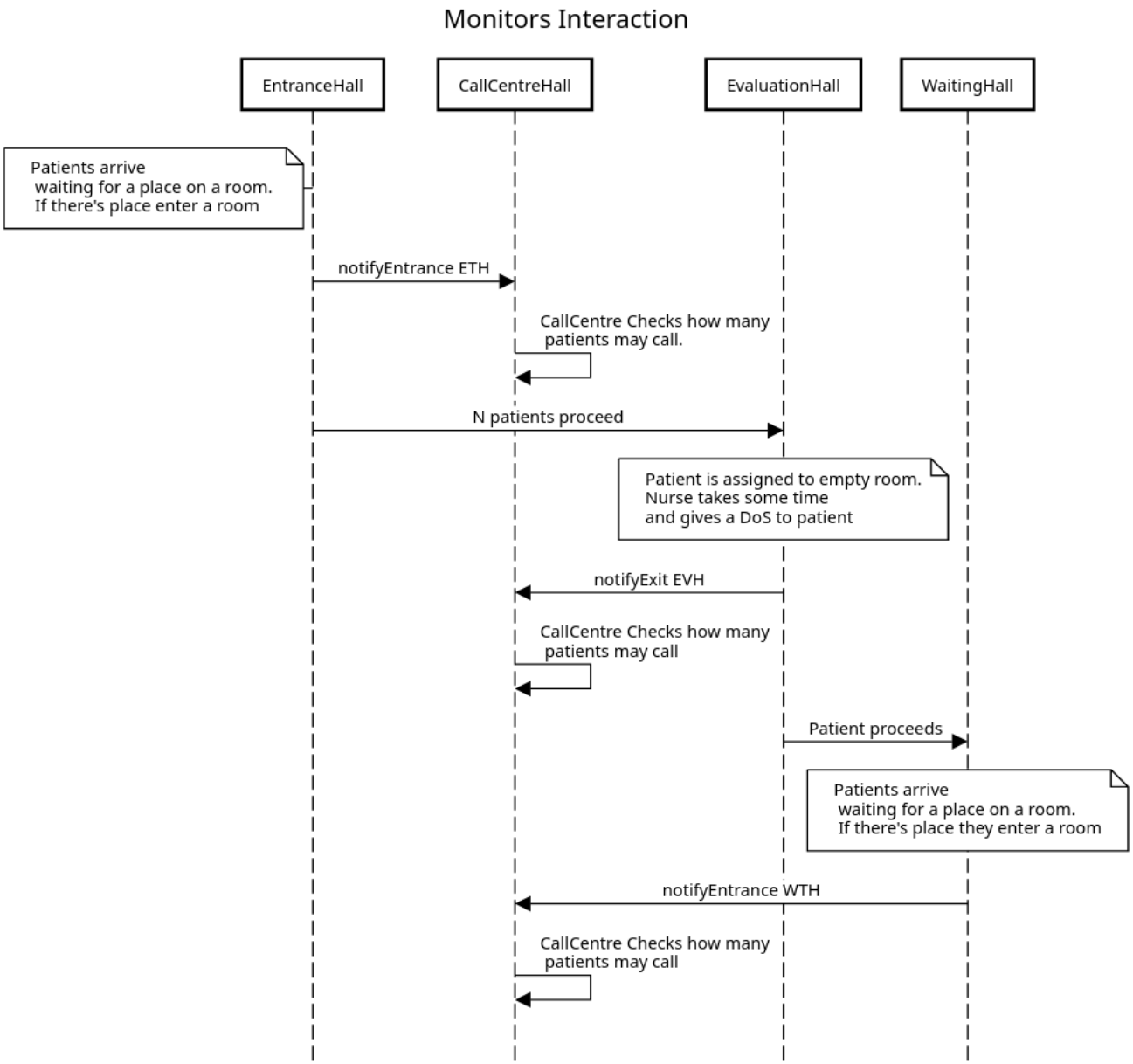

Figure 4: Monitors Interaction Diagram

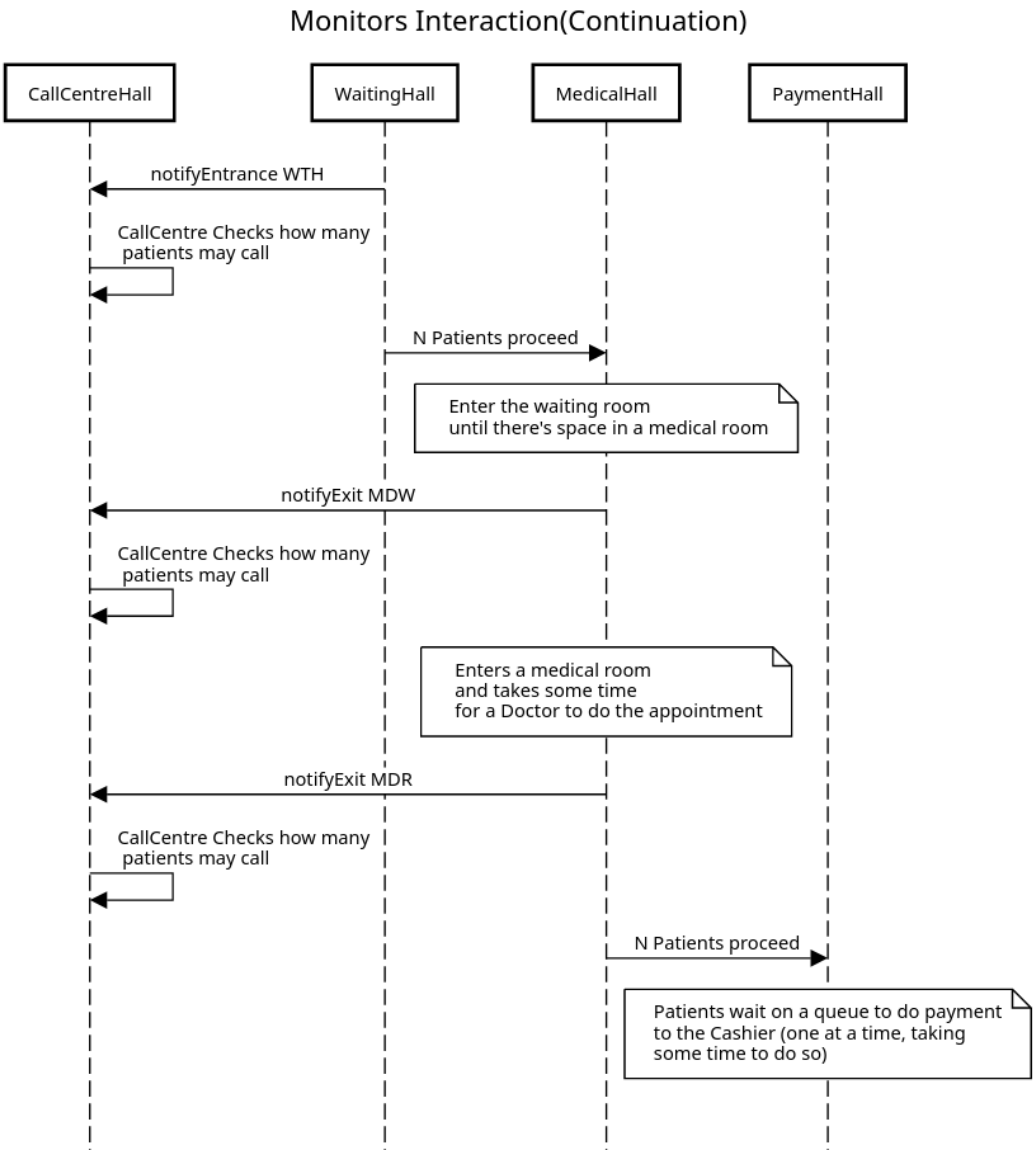

Figure 5: Monitors Interaction Diagram(Continuation)

5.3 Locking Mechanism Implemented

As it as requested for this assignment, ***Reentrant Lock*** was used. The use of the *Reentrant Lock* was really useful to avoid running conditions in the Shared Region of the monitors. Thus, every situation in the monitors that required variables that could be affected by a running condition was always protected by a *lock*. Besides this, the use of *Condition* variables, with a respective *Boolean* flag, were fundamental to take control of threads: whether they should stay blocked or proceed. For instance, in great part of the monitors *Priority* Queues were used, as mentioned previously. For each Priority Queue, it was necessary:

- An array of boolean flags, to control the exit of threads, with the size of the number of threads (patients)

allowed which helps to control specific threads

- The respective array of Conditions, with the size equal to the number of threads.
- A condition and a boolean flag to check if the priority queue was full or not.

6 Work Contribution

For this assignment, the contribution of each student was equal, 50%.

7 Conclusion

With this work, we managed to get a deeper understanding over concurrency and the locking mechanisms. We believe that we reached the main goals of the assignment, by implementing all desired functionalities, following the necessary requirements.