Specification design validation

Report

Alexis DOS SANTOS - Frédéric ROSSO

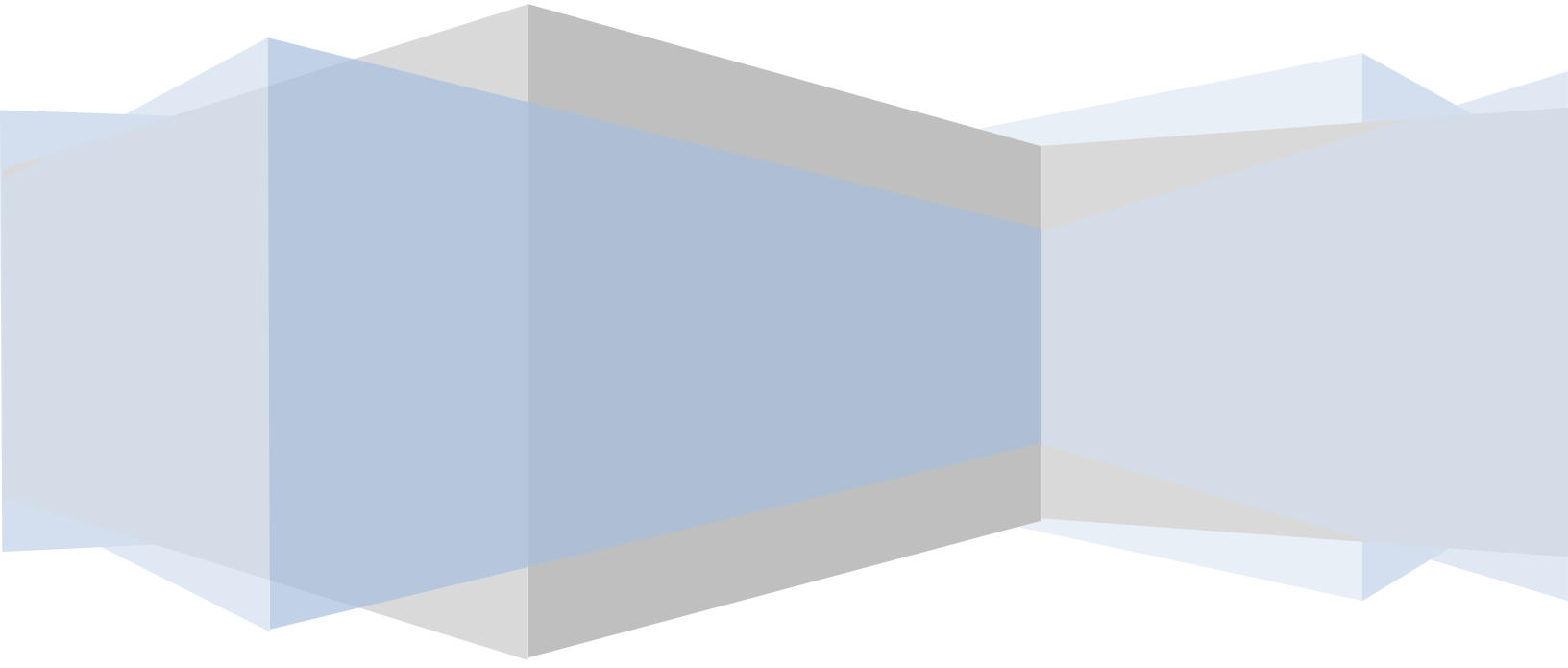


Table of contents

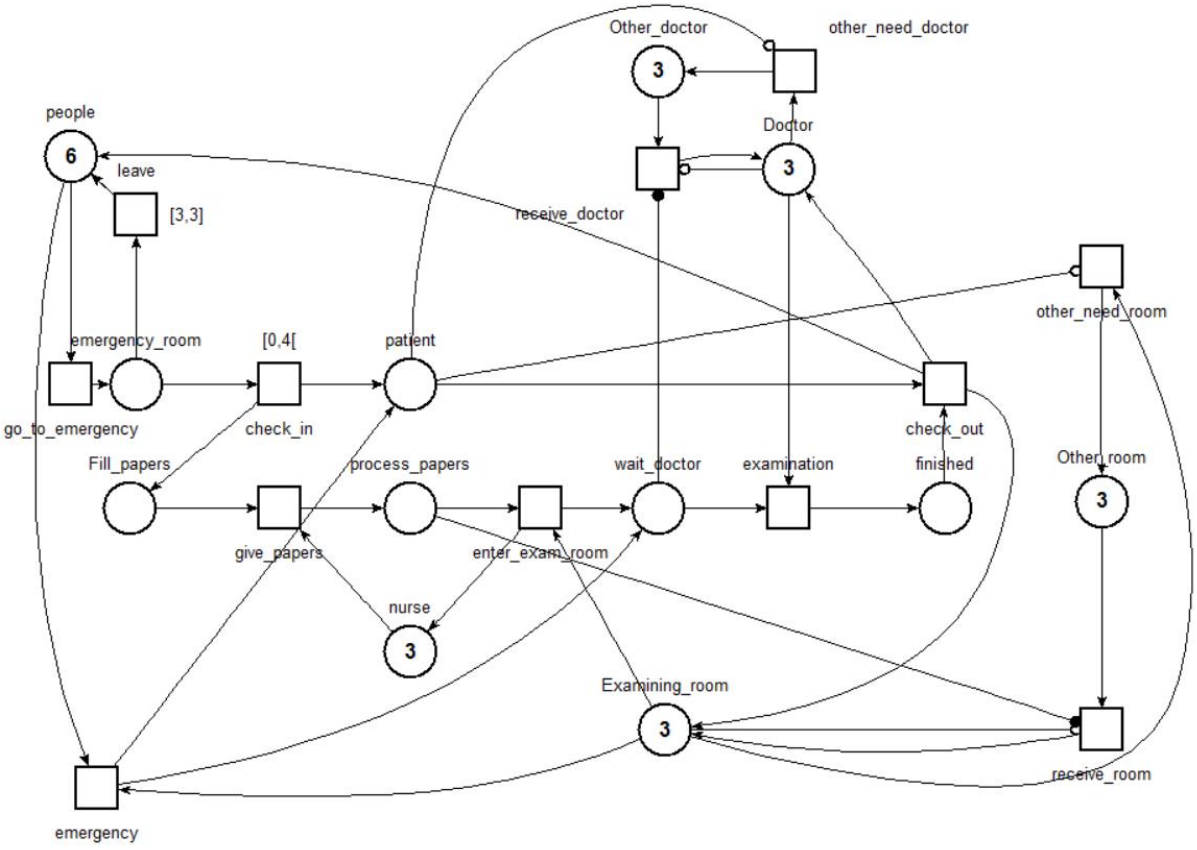
Formal modeling and verification

We did the petri net to model the system. Firstly, we made the emergency care service:

- People enter the hospital thanks to emergency room (the receptionist);
- Depending on the time, the receptionist accepts or refuses the person;
- Patient check in and fill papers;
- A nurse processes them and the patient enter in an examining room;
- The patient is waiting a doctor;
- Then, the doctor examines him;
- The patient checks out.

If it's a gravely injured people, he/she go directly to the examining room.

After that, we made the resource provider, that allows to ask and give doctors or rooms to another service. Thanks to an inhibitor arc, we can know if there is no doctor into the service. So, we can ask for to another service. It's the same way to process for the rooms.



Petri net

We checked the deadlock freeness of our petri net: we check that a patient will always leave the service to return into people (to come back to the beginning). A patient is in the hospital, people is outside.

```
- [](patient => <> people);
TRUE
0.000s
```

We also check when a patient is admitted in the service, he/she will eventually be examined by a doctor. Fill_papers is just after the check in, finished is when the doctor has just examined us.

```
- Fill_papers => <> finished;
TRUE
0.000s
```

Finally, we check if each time an examining room is booked, it will eventually be released. Wait_doctor is when we are in the room waiting doctor, and people is outside the hospital.

```
- [](wait_doctor => <> people);
TRUE
0.000s
```

UML-based modeling

We made the three UML asked in the subject.

Use case diagram

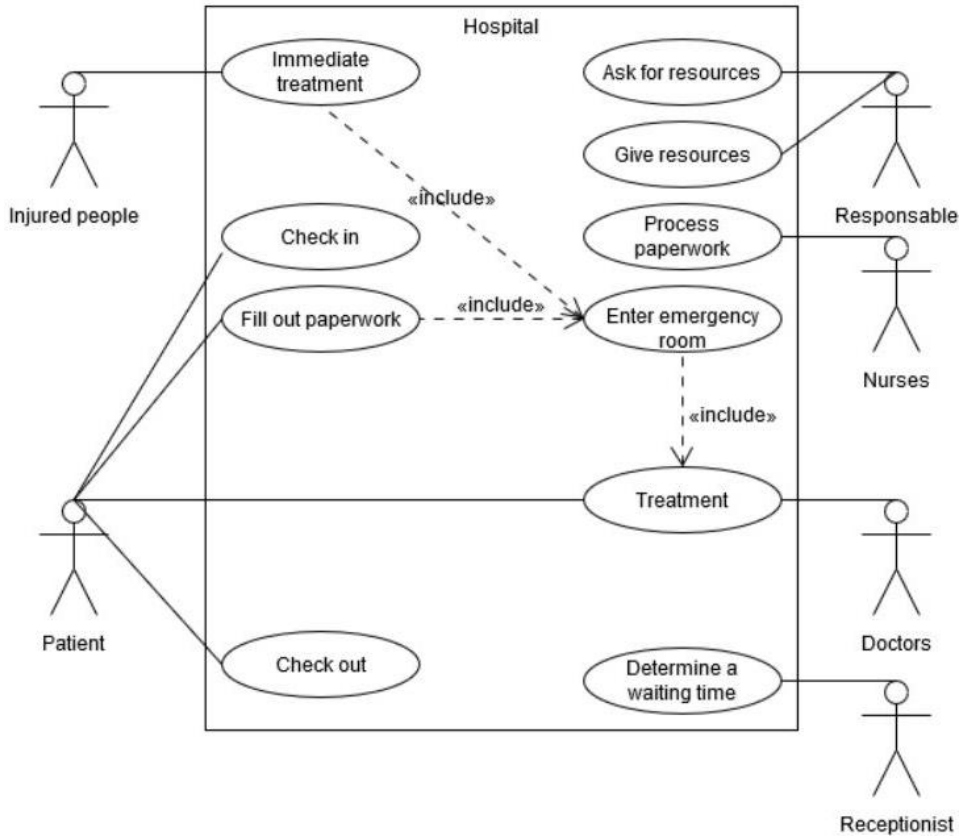
We have two types of patients:

- Injured people: that go directly into the emergency room;
- Normal patient: that need to check in, fill papers, wait into the emergency room, do treatment, and finally check out.

In the hospital, we have different actors:

- Receptionist: he/she only determines the waiting time to know if the patient will be accepted;
- Doctor: he/she just examine the patient;
- Nurse: he/she processes the papers;
- Responsible: he/she manages the resources (ask and give).

Each of these actors has to do some actions define in the subject, like this:



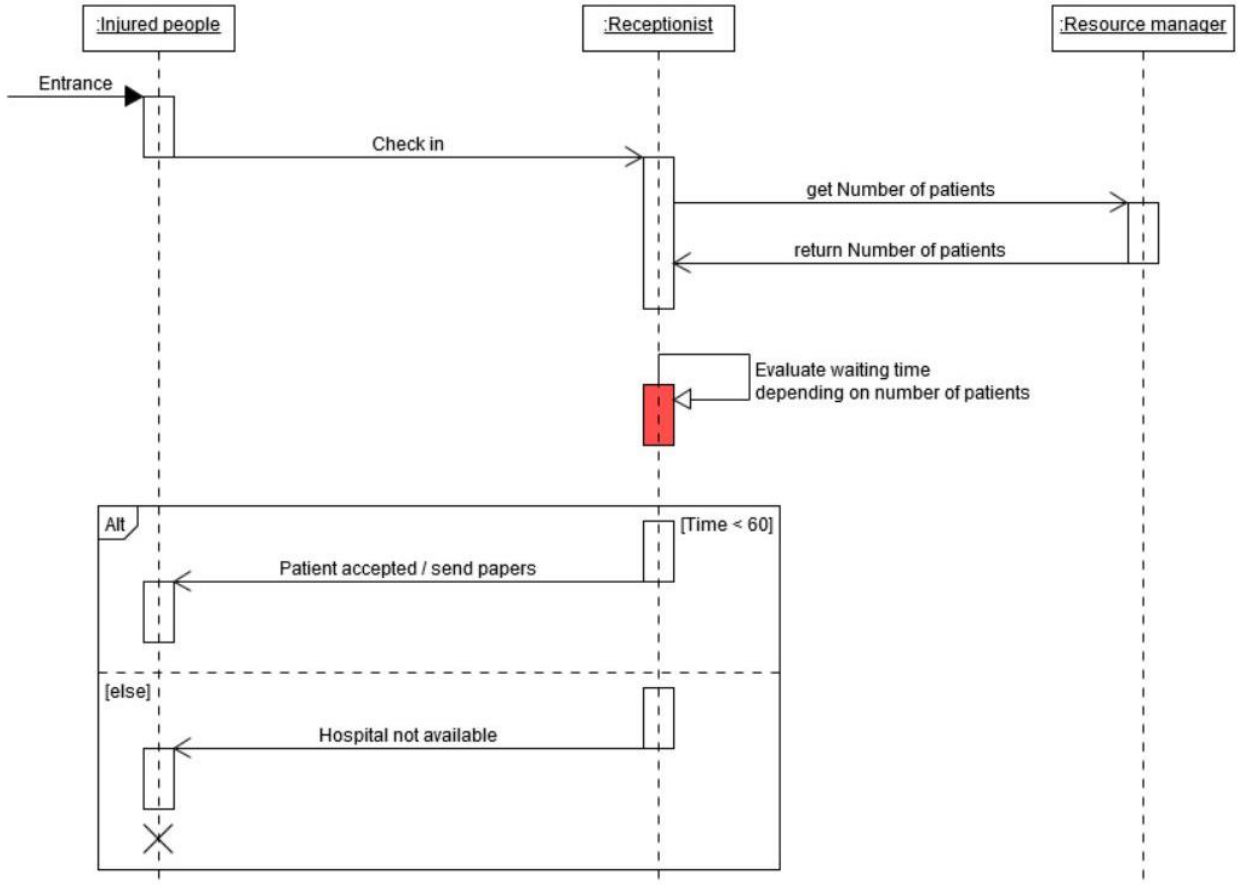
Use case diagram

Sequence diagram

We study the sequence diagram representing the admission of a simple patient in the emergency care service. So, the patient checks in with the receptionist.

The receptionist will ask to the resource manager the number of patients into the service. Depending to the resources (doctors), if there are more patients in the service than doctors, the new patient will be refused at the beginning. If it's ok, he/she can enter.

It's the receptionist that evaluate the waiting time and accept or refuse him.



Sequence diagram

Class diagram

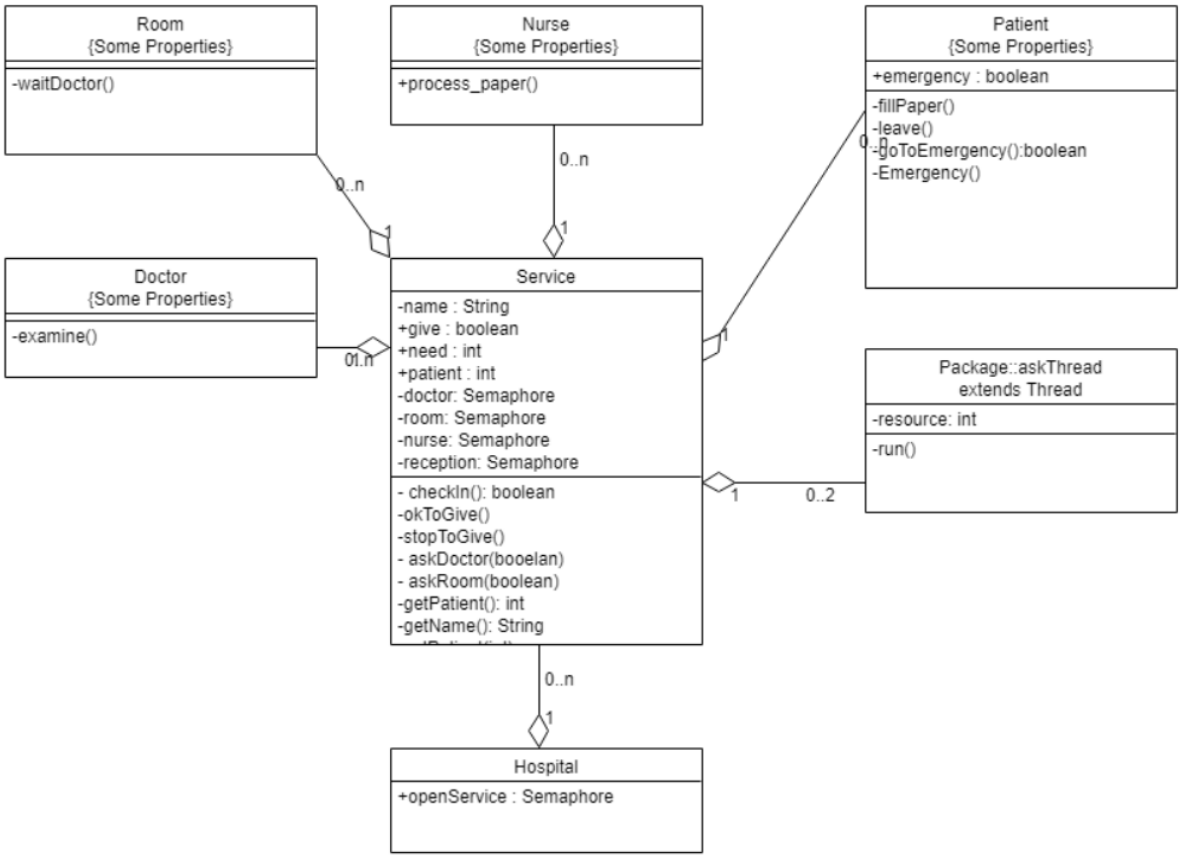
We made a hospital class with one attribute: a semaphore to represent the service which can give a resource. A hospital has many services whereas a service belongs to only one hospital.

A service has many actors:

- A doctor class: the doctor examines the patient;
- A room class: the room needs to wait a doctor;
- A nurse class: he/she processes the paper;
- A patient class: he/she fill papers, go to emergency (to check in), to leave (check out) and emergency (for gravely injured people: the Boolean).

Finally, the service class contains the entire logic of the program. Each service has a name to differentiate them. A give boolean to know if the service is able to give doctors or rooms (or both of them: we know that thanks to the value of integer variable: need). “Patient” represents the number of patients into the service. And, the most important part: to represent each resources (doctors, nurses, rooms and the receptionist), we use semaphores of a determined size.

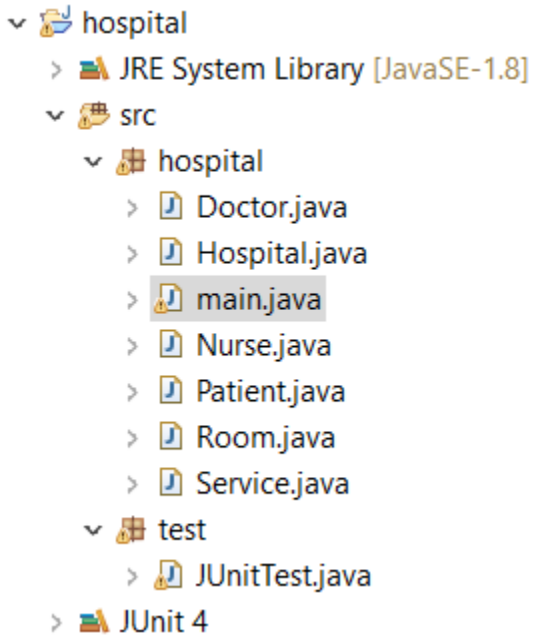
So, for example, when a room needs a doctor to do the examination, it simply acquires one doctor from the semaphore.



Class diagram

Java code

The java code follows the class diagram.



And when a patient comes in service (you can test by launching the main and putting true in the boolean "test") it gives this scenario.

```
Cardiology is ready
Neurology is ready
Rheumatology is ready
A patient is comming in Cardiology
check in ok, Cardiology
He filled the paper,Cardiology
a nurse is processing paper
a nurse is waiting a room,Cardiology
She has finished,Cardiology
The patient wait the doctor in the room,Cardiology
the doctor is here,Cardiology
He finished to examine,Cardiology
The patient leave Cardiology
7.631999146 ms
```

The patient works like a thread and use the different ressources of the service to continue his visit.

Each task takes times, like Fill_papers(), process_paper() and examine(), to simulate the action.

To ask some ressources is also in a thread to let the service works while he is waiting the ressource.

JUnit test

In the test we do the same as the patient thread, we simulate it, in different validation, unit and integration test.

We have 7 different scenarios:

- ServiceTest

  We check that the Service class works well.

- PatientTest

  We check that the Patient class works well.

- Test

  It is the simpliest test, we try how the code works when one patient come in a normal way.

  We check that he is in the service, and that the service ressources are used and released as the next tests.

- RealPatientTest

  Same test than the previous one, but with the patient thread: we test the patient.

- EmergencyTest

  Same as before but when the patient comes in emergency.

- GiveDoctorTest

  Here our service can give a doctor and another one want one, and we check that we cannot give it yet if a patient is coming in our service. And of course that at the end, our service gives the doctor and the other receives it.

- AskRoom

  We want a room, we check that there is no problem if a patient comes and then still wait the room while we stop to ask or another service can give, here, another gives at the end.

- NotEnoughRoom

  We fulfill the room and check that the semaphore is full and that other patients have to wait until one is free

- HasToLeave

We check that the check in can ask a patient to leave the hospital if there is too many patients