Modeling Languages Project Report

Juan Jose Soriano Escobar

January 3, 2018

Contents

List of Figures

A Introduction

Modeling is an important part of any System or product development that helps to understand the model system an holistic point of view and to ensure that all the functionalities and stakeholders requirements are cover. One of the most popular modeling tools nowadays in the software and product development environment is UML.

UML is a communication standard for the world of software development. It consists of different types of diagrams that will, together, help you describe the boundary of the system, the structure of the system, and the behavior of the system.

In this report, a Hospital appointment and billing system management are described and modeled in UML as result of the concepts and diagrams learn in the course of Modeling Languages.

The description of the model in this report will be split by diagram types, covering use case, sequence, activity, state and class diagram.

B Use Case Diagram

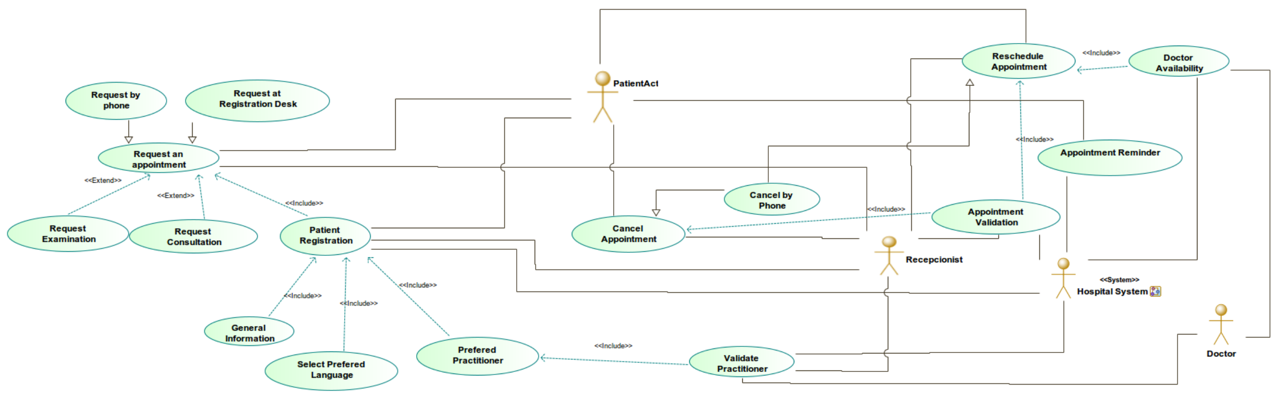
A use case captures a contract between the stakeholders of a system about its behavior describes the systems behavior under various conditions as it responds to a request from the primary actor (Delligatti, Steiner, Soley, 2013).

The case diagram is also good to ensure that every member in the development team understand the main activities and behavior of the system, recognizing the actors and the global interactions between them

The use case diagram is divided in three main behaviors, the appointment management, clinical history management and payments.

B.1 Appointment Use Case

In the figure 1 the use case diagram for the appointment system is Illustrated. It contains four Actors, three Human actors such as the Patient, the Receptionist and the Doctor. The remaining actor is the Hospital System which is consider as a Software from the hospital that is used to manage all the logistics, billing and service operations by different users.



Created with Modelio 3.7.

Figure 1: Appoint Use Case Diagram

This diagram presents how the patient communicates with the receptionist by phone or at the Hospital desk, to Schedule, cancel or modify an appointment. To do so, the Receptionist has to use the system, registering the patient and validating if the doctor is available for the required appointment.

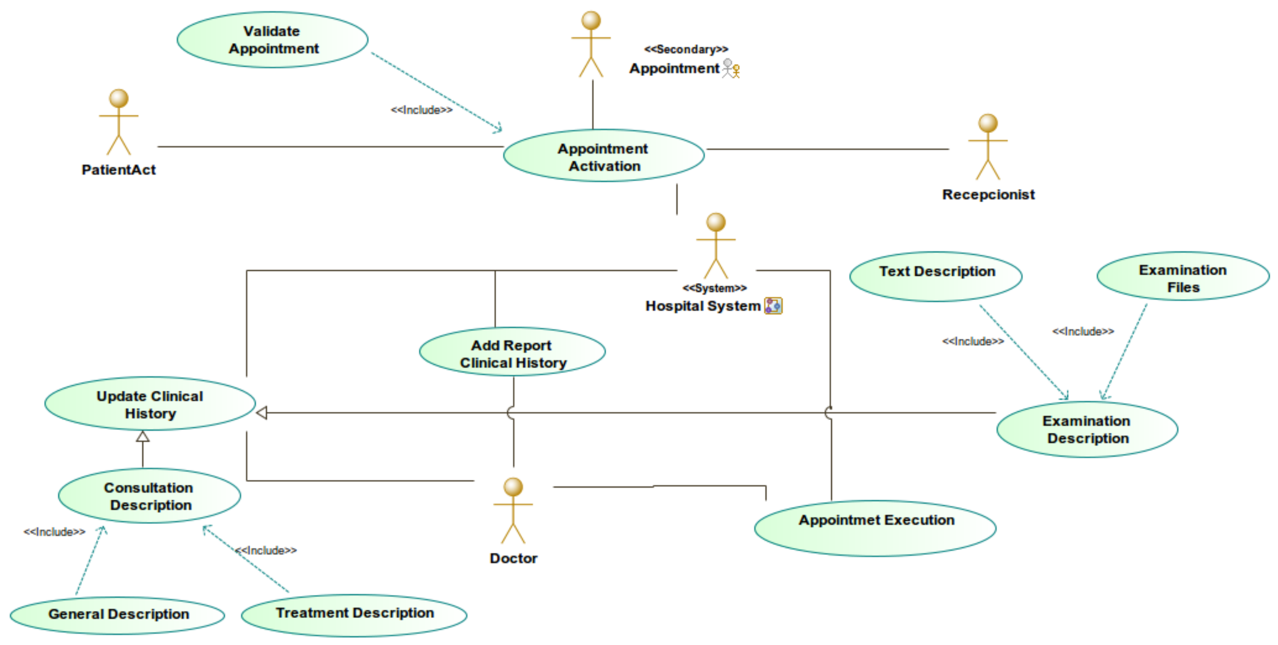
B.2 Examination Use Case

Once the patient has an appointment scheduled, the next step is to understand the appointment behavior. As actors, there are the same four included in the previous Use Case Diagram plus the appointment considered as a secondary one.

The day of the appointment, the Patient will go to the Hospital desk by the time of the appointment as is expected. The receptionist will activate the appointment in the system, to let know to the Doctor and to the billing staff that the patient has come to the appointment as planned.

Once in the execution of the appointment with the Doctor, depending on the type of the appointment (Examination or consultation), the Doctor can rad and Update the Patient Clinical History by using the Hospital System. During the reportm the Doctor is allowed to add files, treatment description and some text explaining the patient condition.

Other interaction between the Doctor and the system such as creating reports and email notifications will be explained later in the Sequence diagram.

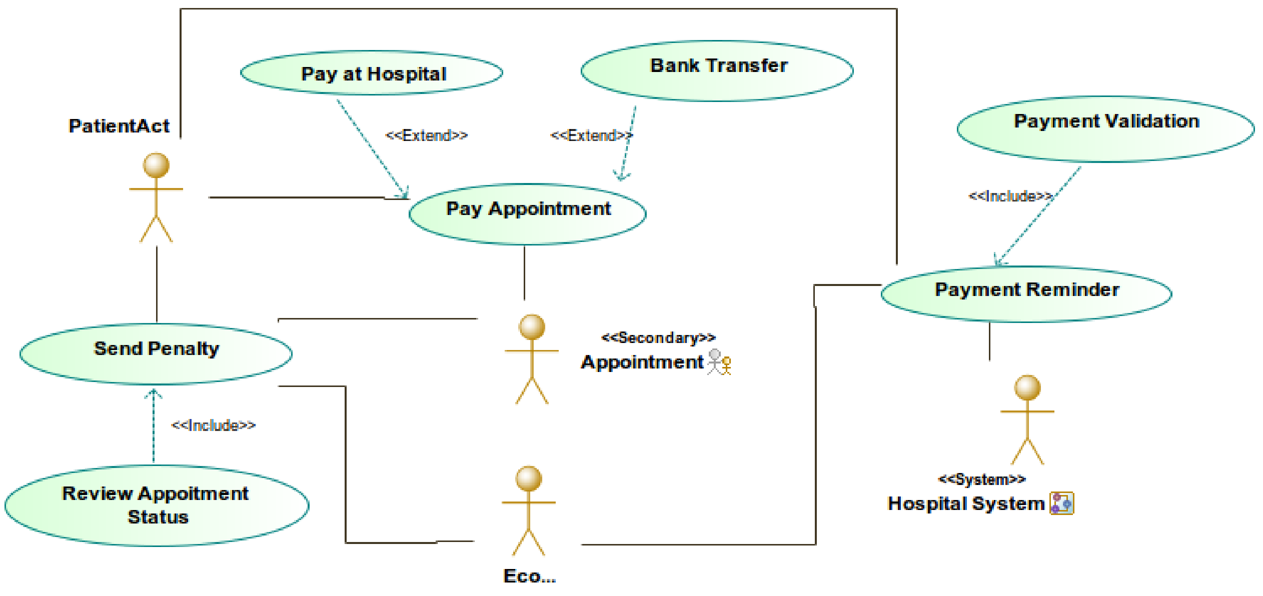


Created with Modelio 3.7.

Figure 2: Examination Use Case.

B.3 Payments Use Case

In the payments Use Case, the appointment becomes a secondary actor by itself because the appointment is the connecting factor (somehow the product consume by the patient) between the Patient and the Hospital. In the diagram included in the Figure 3, the Economy staff Actor from the hospital is introduced to review the payment status of the appointments and create penalty fees whenever is needed. The Economy staff uses the Hospital system to access to the appointments an patient payments.



Created with Modelio 3.7.

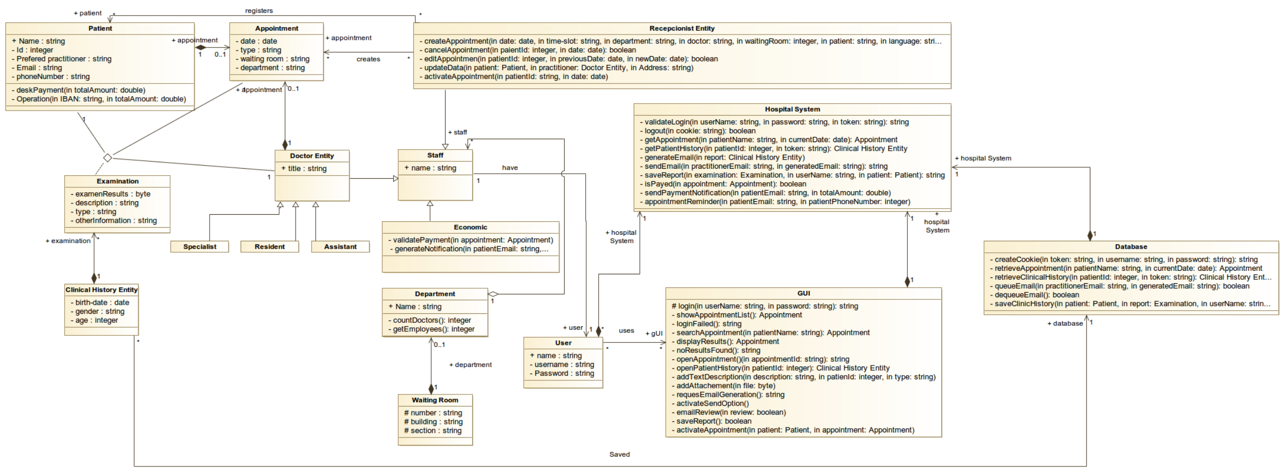
Figure 3: Payment Use Case Diagram

The Patient have the option of paying at the hospital or by a bank transfer.

C Class Diagram

Class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modeling of object oriented systems because they are the only UML diagrams, which can be mapped directly with object-oriented languages.

The class diagram explains in detail the entities with it respective attributes and operations available. It also describes the relation, multiplicity and association between the the different elements. The class diagram is one of the main elements when the system is design and the one that keep updating during the design of the other diagrams.



Patient
+ Name : string
- Id : integer
- Prefered practitioner : string
- Email : string
- phoneNumber : string
- deskPayment(in totalAmount: double)
- Operation(in IBAN: string, in totalAmount: double)
Appointment
- date : date
- type : string
- waiting room : string
- department : string
Recepcionist Entity
- createAppointment(in date: date, in time-slot: string, in department: string, in doctor: string, in waitingRoom: integer, in patient: string, in language: stri...
- cancelAppointment(in paientId: integer, in date: date): boolean
- editAppointmen(in patientId: integer, in previousDate: date, in newDate: date): boolean
- updateData(in patient: Patient, in practitioner: Doctor Entity, in Address: string)
- activateAppointment(in patientId: string, in date: date)
Hospital System
- validateLogin(in userName: string, in password: string, in token: string): string
- logout(in cookie: string): boolean
- getAppointment(in patientName: string, in currentDate: date): Appointment
- getPatientHistory(in patientId: integer, in token: string): Clinical History Entity
- generateEmail(in report: Clinical History Entity)
- sendEmail(in practitionerEmail: string, in generatedEmail: string): string
- saveReport(in examination: Examination, in userName: string, in patient: Patient): string
- isPayed(in appointment: Appointment): boolean
- sendPaymentNotification(in patientEmail: string, in totalAmount: double)
- appointmentReminder(in patientEmail: string, in patientPhoneNumber: integer)
Doctor Entity
+ title : string
Staff
+ name : string
Examination
- examenResults : byte
- description : string
- type : string
- otherInformation : string
Specialist
Resident
Assistant
Economic
- validatePayment(in appointment: Appointment)
- generateNotification(in patientEmail: string,...
Clinical History Entity
- birth-date : date
- gender : string
- age : integer
Department
+ Name : string
- countDoctors(): integer
- getEmployees(): integer
Waiting Room
# number : string
# building : string
# section : string
User
+ name : string
- username : string
- Password : string
GUI
# login(in userName: string, in password: string): string
- showAppointmentList(): Appointment
- loginFailed(): string
- searchAppointment(in patientName: string): Appointment
- displayResults(): Appointment
- noResultsFound(): string
- openAppointment()(in appointmentId: string): string
- openPatientHistory(in patientId: integer): Clinical History Entity
- addTextDescription(in description: string, in patienId: integer, in type: string)
- addAttachement(in file: byte)
- requesEmailGeneration(): string
- activateSendOption()
- emailReview(in review: boolean)
- saveReport(): boolean
- activateAppointment(in patient: Patient, in appointment: Appointment)
Database
- createCookie(in token: string, in username: string, in password: string): string
- retrieveAppointment(in patientName: string, in currentDate: date): Appointment
- retrieveClinicalHistory(in patientId: integer, in token: string): Clinical History Ent...
- queueEmail(in practitionerEmail: string, in generatedEmail: string): boolean
- dequeueEmail(): boolean
- saveClinicHistory(in patient: Patient, in report: Examination, in userName: strin...
+ patient
registers
+ appointment
creates
+ staff
have
+ hospital System
+ examination
+ department
+ user
uses
+ gUI
+ database
Saved

Created with Modelio 3.7.

Figure 4: Class Diagram.

In the figure 4 the class diagram for the hospital application is presented. All the Actors in the use case diagram became entities and some sub-classes are derive from a main class. A good example is the Staff class that includes all the personal working at the Hospital in different departments. At the first level is divided in Receptionist, Doctor Entity and Economic entity, and at the same time, the Doctor entity is generalize into Specialist, Resident and Assistance. The practitioner is considered as any of the Doctor entities that is assign to a patient.

Additionally, the staff is associated to a Department and every Department contain waiting rooms that are addressed by a number, building and Section.

As it was mentioned before, the hospital system is considered as a local software that manages the appointments and the clinic histories of the patients. In oder to access to the sytem, every member of the staff have an account with it respective user name and password. Inside the system the user interacts with a Graphical User Interface that displays the appointments, allow the receptionist to validate and schedule the appointments and the doctors to read and write the medical history of the patient.

The system also generates and schedule notifications. I like to imaging the system as a server running all the logic of the application, connected to a Database where all the Clinical Stories, Users and bills are saved. The GUI is part of the front-end (and middleware) of the application and the system is the back-end, using a **token** to authenticate and establish the connection with the database.

It is easy to identify the system controller (application) as the entity with the biggest amount of operations. This is due to its "gateway" nature to allow the interaction between the entities and actors in the system.

Some of the main functions are:

C.0.1 GUI operations

- **Login(String username, String password):String =** It allows the user to authenticate to the system and access to the Patient, appointment and billing information. This method returns an String with the new connection view for the user. It is declare as an String but it could also be a XML or HTML file for a web application. The login connects to the system and use the function *validateFunction*.
- **loginFail():String =** is an operation that is triggered when the user credentials don match with the ones created in the system.
- **searchAppointment(String patientName):Appointment =** is the function use by the doctor to search the scheduled appointments by patient name. It will return the appointment(s) as an Appointment object that would be displayed in the GUI.
- **displayResults():Appointment =** display in the GUI the searched appointment.
- **openAppointment(String AppointmentId):String =** request to the system the details of the appointment with an specific Id to later display them in a view.
- **addTextDescription(String decription, int patientIdm String type):String =** this operation receives the type of the appointment of the patient, and allow the doctor to write a description in the clinic history of the given patient.
- **addAttachement(byte file):void =** attaches to the report of the appointment a string of bytes that could include images, PDF files or any other kind of document.
- **requestEmailGeneration():String =** request to the system to queue and email with the report fir the general practitioner of the patient. This operation does not include the patient data because I assume the system already have loaded the patient data during the appointment execution. It returns the ready or fail generation message.
- **activateSendOption():void =** prior to the email generation, the doctor has to activate and allow the send option during the report.
- **emailReview(Boolean review):void =** once the email is generated in the preview version of it is loaded, the Doctor had the to approve and definitively send the queued mail.
- **saveReport():Boolean =** save the current state of the appointment report in the clinical history. Returns true when it is successfully saved or false when there is any inconsistency.
- **activateAppointment(Patient patient, Appointment appointment) =** this function is used in the GUI to start the appointment and let know to the billing department that the patient attended to the schedule appointment.

C.0.2 Hospital System operations

- **validateFunction(String username, String password, String token):String =** This function receives the username and the password from the GUI and adds the unique token to access and validate the user in the database. This method will return a string with the view and the user cookie. The cookie is assigned by the database with the function *createCookie*.
- **logout(String cookie):boolean =** destroy the cookie created for the user access.
- **getAppointment(int patientId, Date currentDate):Appointment =** receives form the GUI the patientId and adds the current date to search for the available appointments of the given patient in the present day.
- **getPatientHistory(int patientId, String token):Clinical History Entity =** receives from the GUI the patientId and adds the access token to retrieve the patient clinical history to the doctor.
- **generateEmail(Clinical History Entity report):void =** consolidates all the information of the appointment and queue an email to the patient practitioner.
- **sendEmail(String practitionerEmail, String generatedEmail):String =** send the report from the system to the patient practitioner.

- **IsPayed(Appointment appointment):boolean =** mainly used by the Economic entity to know if the patient payed the appointment bill.
- **sendAppointmentNotification(String patientMail, Double totalAmount):void =** send the remainder to the patient to pay the bill when is issued as a bank transfer or any other fee.
- **appointmentReminder(String patientEmail, int patientPhoneNumber) =** automatically sends and appointment notification by email and phone to the patient, 24 hrs before the appointment.

C.0.3 Database operations

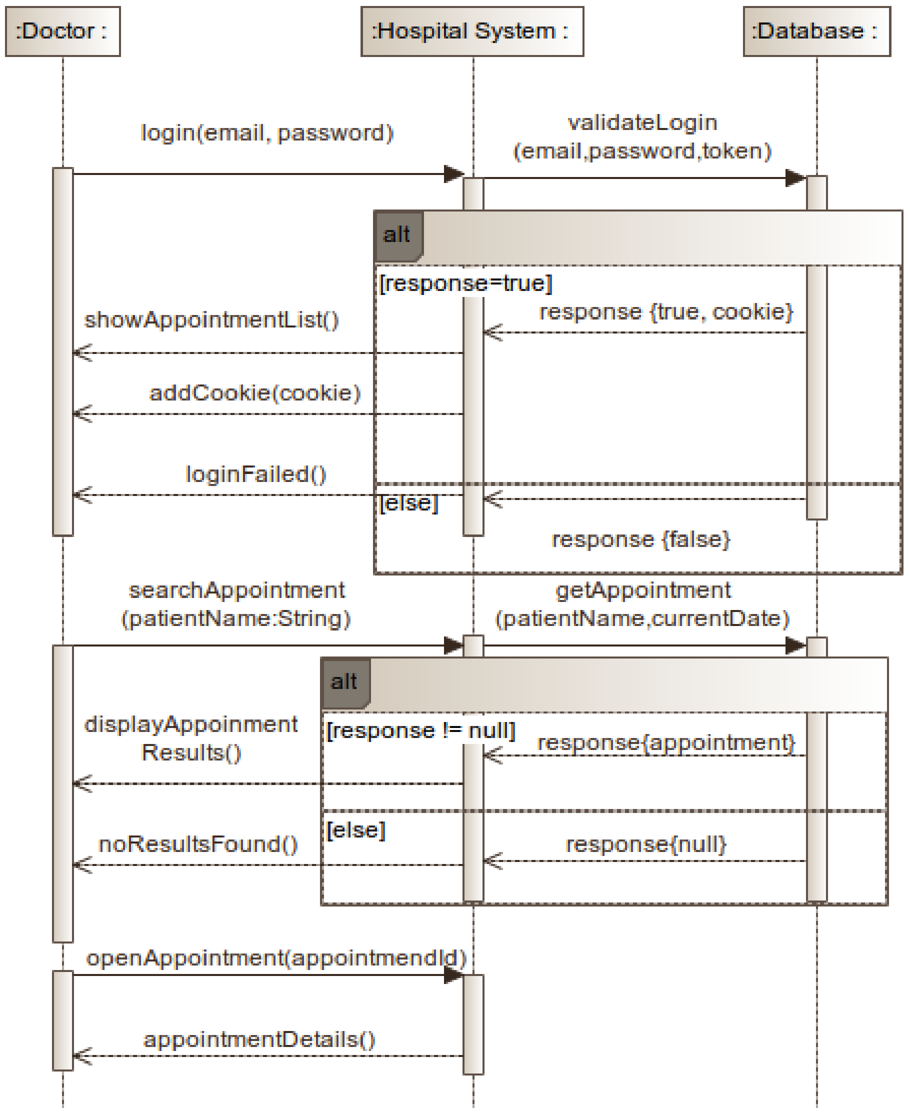
- **createCookie(String token, String username, String password): string =** the database system receives the server token together with the username and password and if the data is correct, it will return a access cookie that is saved and assigned to the user in the database. From this point in advance I assume and is that the cookie is included by the browser or GUI in every operation while the user remains connected and authenticated in the system.

Other operations used by the other entities are not described because its name explains it functionality good enough or is mentioned in other diagram.

D Sequence Diagram

The sequence diagram implement is the appointment execution, involving the doctor, the hospital system and the database. The sequence diagram include some of the operations presented in the class diagram and shows entities interactions arranged in time sequence. As an important remark, I am considering that the Doctor is using an *stateless* web application to access to the hospital system. This also explains why every action ends after the *server* and *database* response.

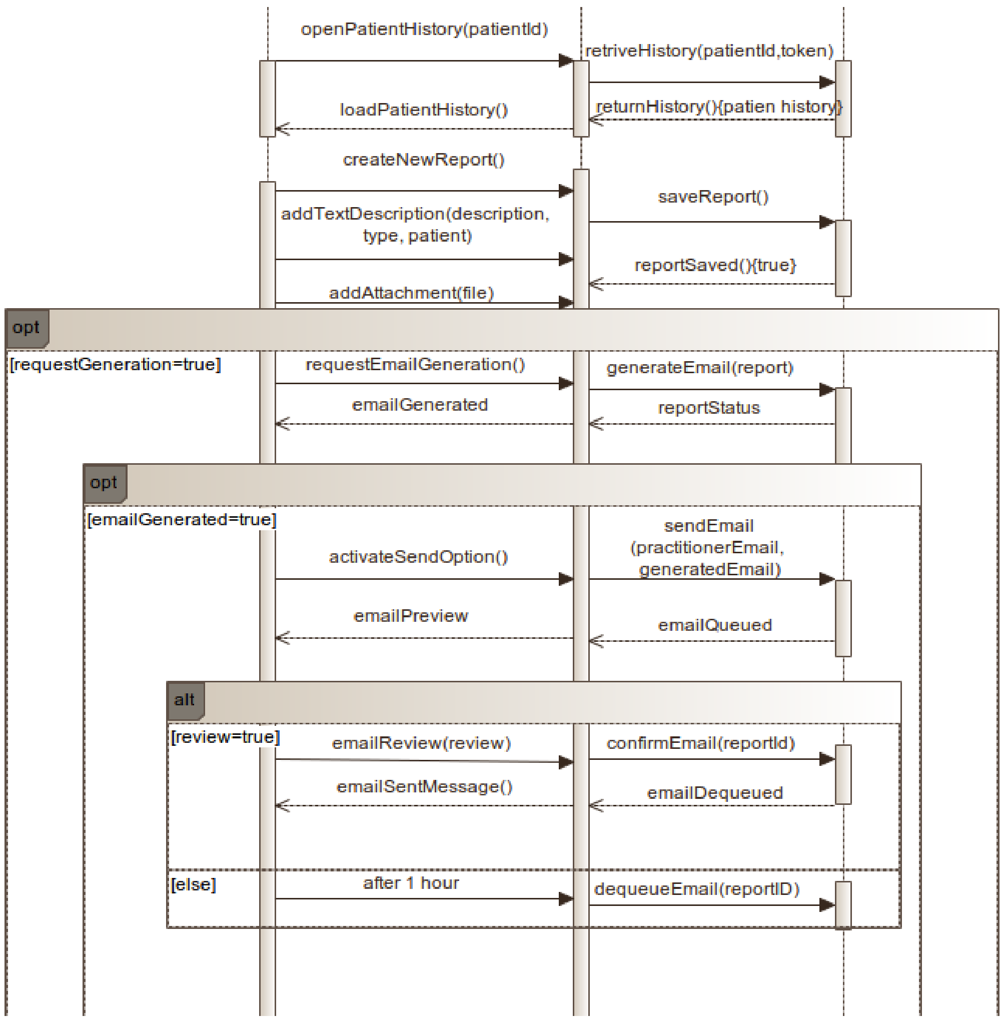
The sequence diagram designed is vertically too long to be included in a single image. For this reason it was split in the figure



Created with Modelio 3.7.

Figure 5: Sequence diagram part 1.

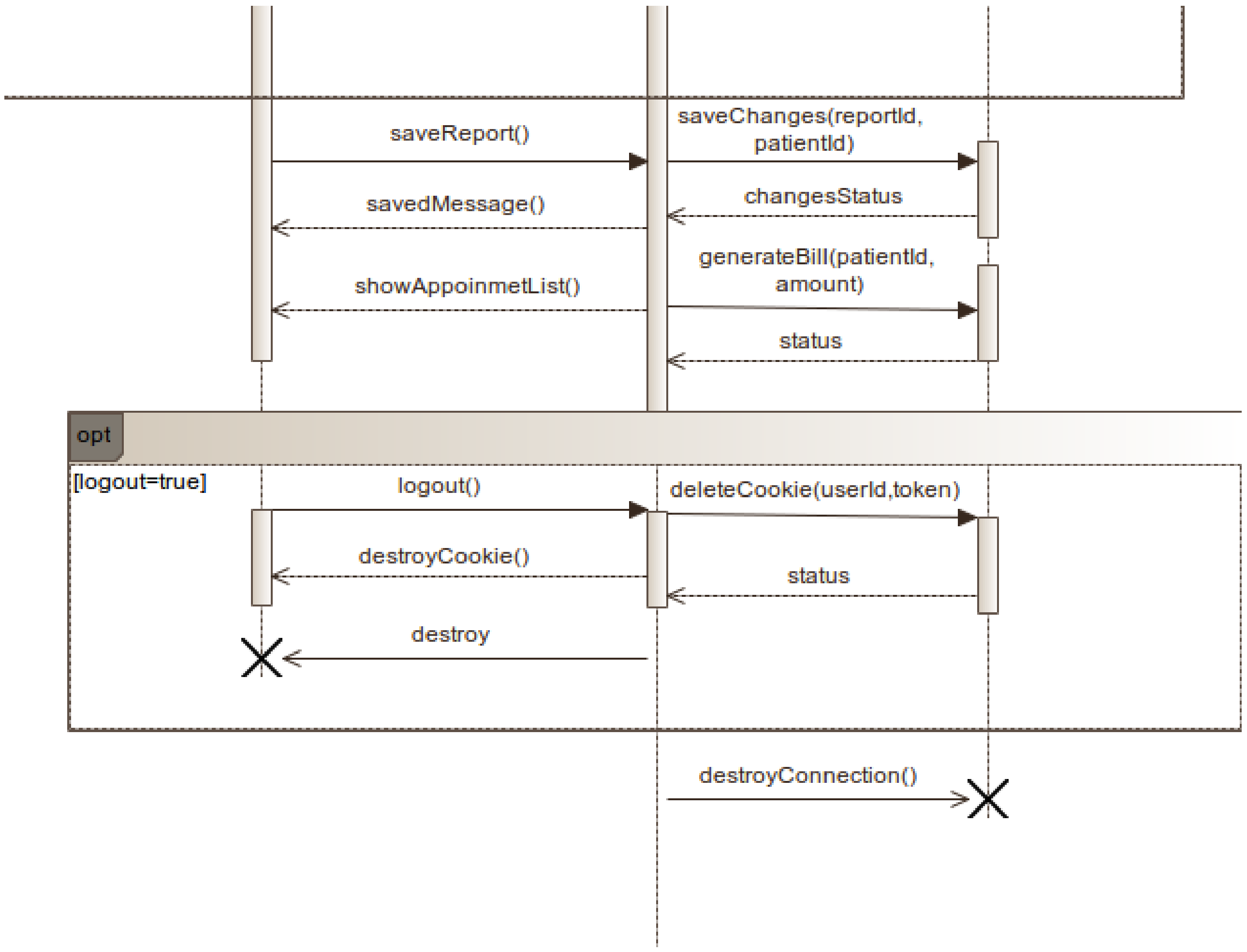
This first part describes the Login and authentication of the Doctor as an user in the hospital system and how the Doctor search for the appointment details.



Created with Modelio 3.7.e

Figure 6: Sequence diagram part 2.

Continuously the Doctor creates the report by adding descriptions, files with the results or any treatment description. At the end of the report, the doctor has the option of generate, preview and send the report of the appointment to th patient practitioner.



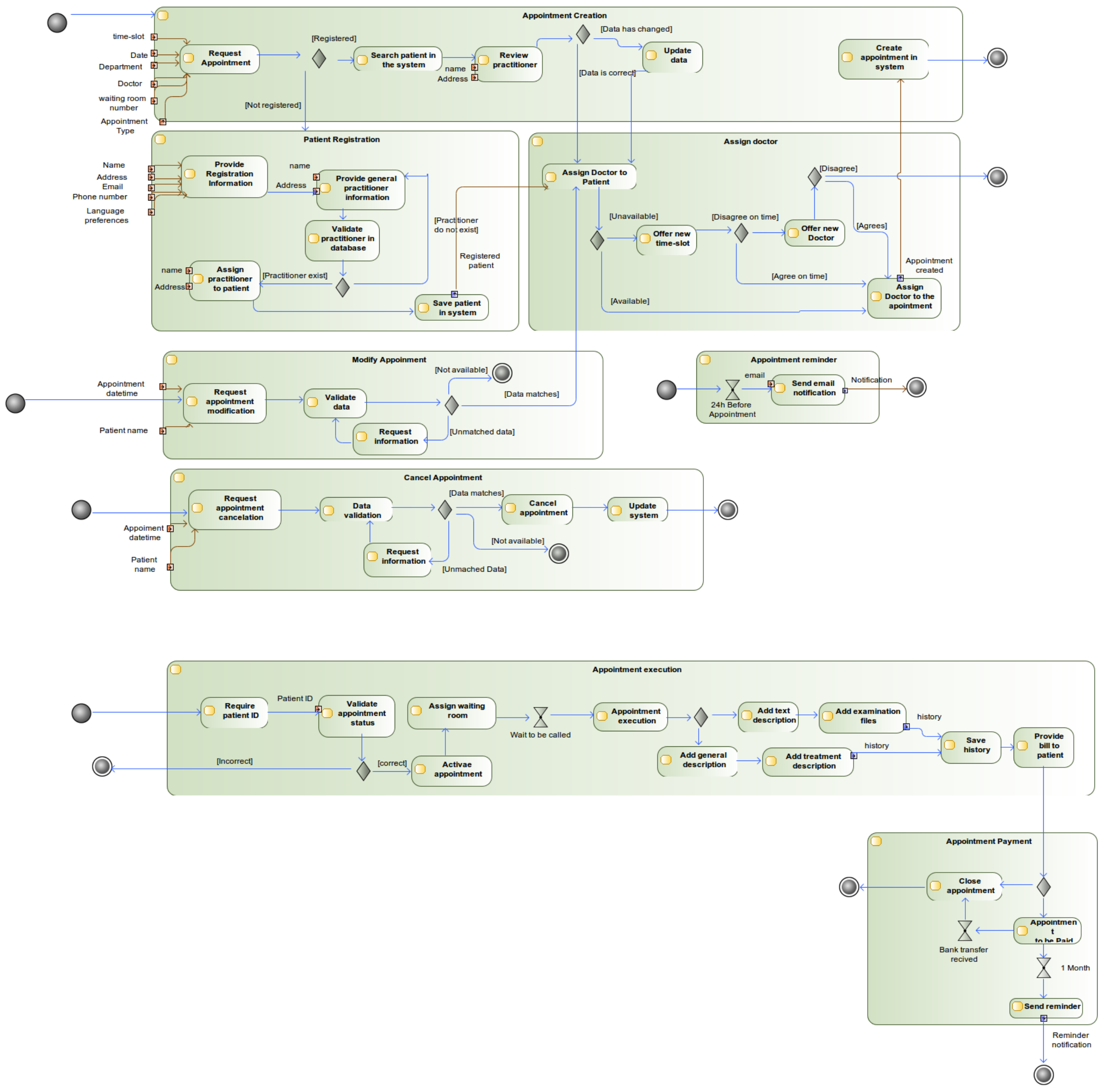
Created with Modelio 3.7.

Figure 7: Sequence diagram part 3.

Finally in the figure 7 illustrates hoe the report is saved in the clinical history of the patient and the creation of the bill to be paid by the patient. At the end of the day the Doctor is able to logout and destroy the user connection with the system.

E Activity Case Diagram

The activity diagrams are dynamic views of the system that expresses sequence of behaviors and event occurrence over time (Delligatti, Steiner, Soley, 2013). Together with the state machine diagram, the system behavior is expressed.

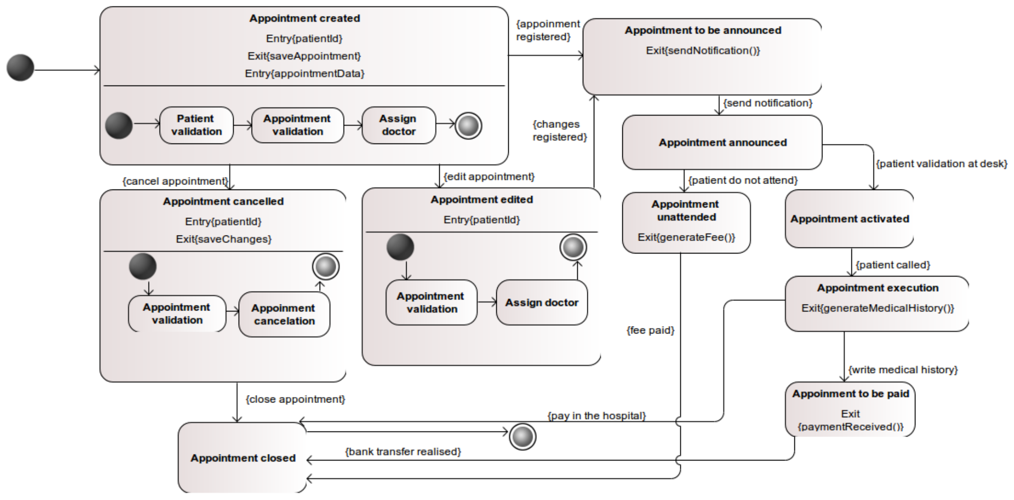


self-made

Figure 8: Distributed Computing Implementation.

F State Diagram

Next to the activity diagram is the State Machine Diagram, which is also a behavior diagram focus on how a structure within a system changes in response to event occurrences over time. It refers to the behavior that begins executing the moment a block is instantiated and generally finishes executing when that instance is destroyed



self-made

Figure 9: Distributed Computing Implementation.