Universidade do Porto

Faculdade de Engenharia

FEUP

# Safety Net Hospital

*Final Report*

Mestrado Integrado em Engenharia Informática e Computação

4ºano

Formal Methods in Software Engineering

**Authors:**

José Martins – up201404189 – up201404189@fe.up.pt

Marcelo Ferreira - up201405323 - up201405323@fe.up.pt

3 de Janeiro de 2018

# Index

# 1. Informal system description and list of requirements

## 1.1. Informal system description

With this system we propose to  model a safety net hospital, in order to achieve this goal the system should be capable of managing all the relevant information related to the safety net hospital network. The following topics describe the main aspects that  we consider relevant and that we will be addressing in our system model:

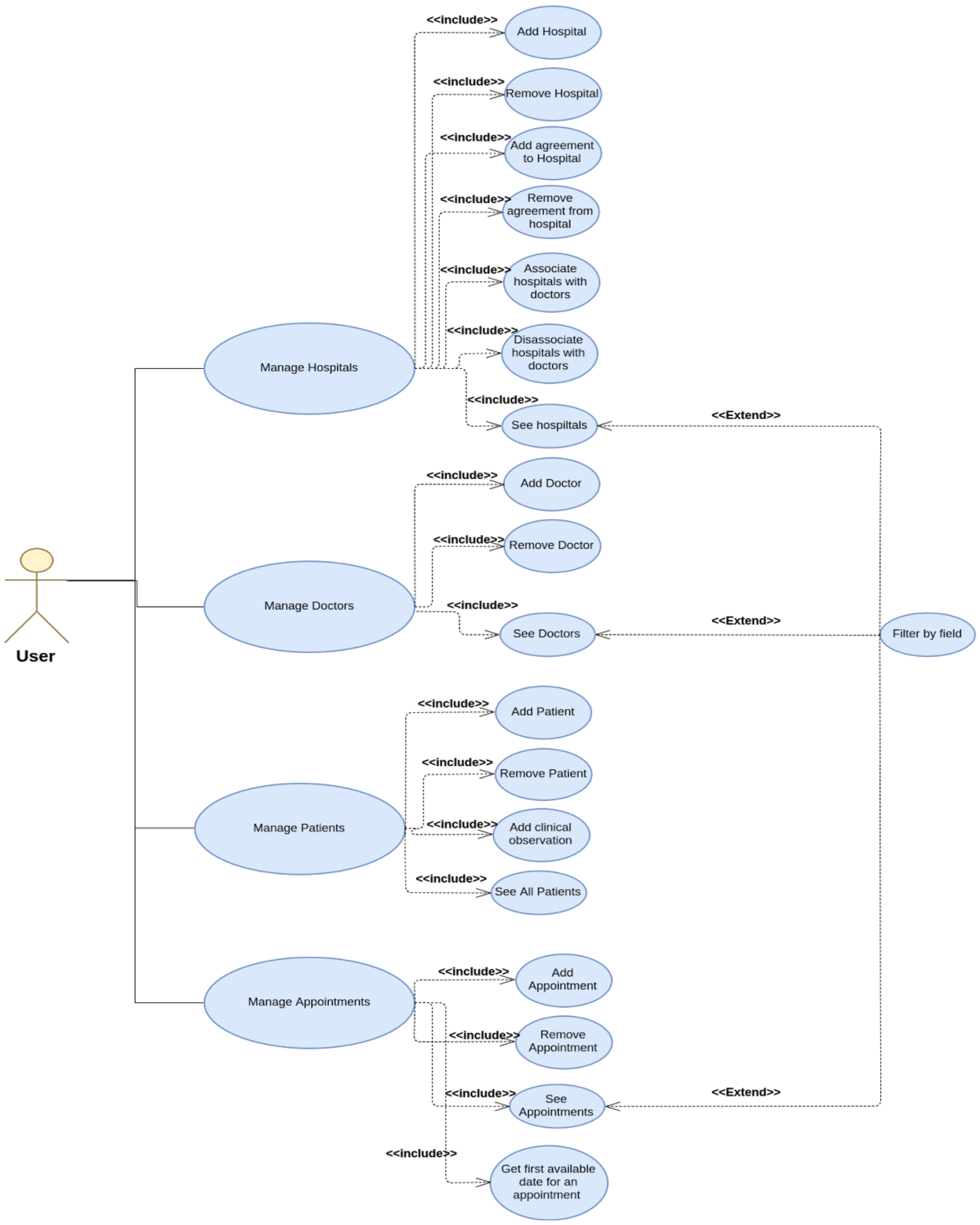
- Add, remove and search hospitals, with the possibility of filtering by name, specialty, agreement and city;
- Add, remove and search doctors, with the possibility of filtering by specialty;
- Add, remove and see patients;
- Add, remove and search appointments, with the possibility of filtering by hospital, doctor and patient;
- Associate/disassociate hospitals and doctors;
- Get the first date available for an appointment, considering parameters like, specialty, hospital and doctors.

## 1.2. List of requirements

| Id | Priority | Description |
|---|---|---|
| R01 | Mandatory | The user shall be able to add and remove an hospital |
| R02 | Mandatory | The user shall be able to see all hospitals and filter them by field (name, specialty, agreement,city) |
| R03 | Mandatory | The user should be able to add and remove a doctor |
| R04 | Mandatory | The user shall be able to see all doctors and filter them by field (specialty) |
| R05 | Mandatory | The user shall be able to add and remove a patient |
| R06 | Mandatory | The user shall be able to see all patients |
| R07 | Mandatory | The user shall be able to schedule an appointment, giving a doctor, hospital, patient and date. |
| R08 | Mandatory | The user should be able to cancel an appointment |
| R09 | Mandatory | The user shall be able to see the network appointments and filter them by (hospital, doctor , patient and specialty) |
| R10 | Mandatory | The user should be able to get the first available date for an appointment, considering parameters like |
| R11 | Mandatory | The user should be able to associate and disassociate a doctor to an hospital |
| R12 | Optional | The user should be able to add medical observations to a patient |
| R13 | Optional | The user should be able to add and remove agreements from an hospital |

# 2. Visual UML model

## 2.1. Use case model



## 2.2. Use cases details

| Scenario | Manage Hospitals |
|---|---|
| Description | Add an hospital to the system |
| Pre-conditions | 1. The hospital can't exist |
| Post-conditions | 1. The hospital exists |
| Steps | 1. Enter hospital name<br>2. Enter hospital location<br>3. Enter hospital address<br>4. Enter hospital post-code<br>5. Add agreement's to the hospital or skip without adding any one |
| Exceptions | (unspecified) |

| Scenario | Manage Hospitals |
|---|---|
| Description | Remove an hospital from the system |
| Pre-conditions | 1. The hospital to be removed has to exist |
| Post-conditions | 1. The hospital can not exist<br>2. There is not any appointment associated with the hospital that was removed |
| Steps | 1. Choose a hospital |
| Exceptions | (unspecified) |

| Scenario | Manage Hospitals |
|---|---|
| Description | Add a doctor to an hospital |
| Pre-conditions | 1. The doctor has to exist on the system<br>2. The hospital has to exist on the system<br>3. The doctor can't be associated to that hospital |
| Post-conditions | 1. The doctor has to exist on the system<br>2. The hospital has to exist on the system<br>3. The doctor is associated to that hospital |
| Steps | 1. Choose a hospital<br>2. Choose a doctor |
| Exceptions | (unspecified) |

| Scenario | Manage Hospitals |
|---|---|
| Description | Remove a doctor from  an hospital |
| Pre-conditions | 1. The doctor has to exist on the system<br>2. The hospital has to exist on the system<br>3. The doctor has to be associated with the hospital |
| Post-conditions | 1. The doctor has to exist on the system<br>2. The hospital has to exist on the system<br>3. The doctor can not be associated to that hospital<br>4. There is no appointment relative to that doctor on that hospital |
| Steps | 1. Choose a hospital<br>2. Choose a doctor |
| Exceptions | (unspecified) |

| | |
|---|---|
| **Scenario** | **Manage Hospitals** |
| **Description** | Add an agreement to an hospital |
| **Pre-conditions** | 1. The hospital exist<br>2. The agreement can not exist on that hospital |
| **Post-conditions** | 1. The hospital exist<br>2. The agreement  exist on that hospital |
| **Steps** | 1. Choose a hospital<br>2. Choose an agreement |
| **Exceptions** | (unspecified) |

| | |
|---|---|
| **Scenario** | **Manage Hospitals** |
| **Description** | Remove an agreement from an hospital |
| **Pre-conditions** | 1. The hospital exist<br>2. The agreement with that hospital exist |
| **Post-conditions** | 1. The hospital exist<br>2. The agreement with that hospital does not exist |
| **Steps** | 1. Choose a hospital<br>2. Choose an agreement |
| **Exceptions** | (unspecified) |

| | |
|---|---|
| **Scenario** | **Manage Hospitals** |
| **Description** | List all the hospitals |
| **Pre-conditions** | N/A |
| **Post-conditions** | N/A |
| **Steps** | N/A |
| **Exceptions** | (unspecified) |

| | |
|---|---|
| **Scenario** | **Manage Hospitals** |
| **Description** | List all the hospitals filtered by field |
| **Pre-conditions** | N/A |
| **Post-conditions** | N/A |
| **Steps** | 1. Enter the value of the field |
| **Exceptions** | (unspecified) |

| | |
|---|---|
| **Scenario** | **Manage Doctors** |
| **Description** | Add a doctor on the system |
| **Pre-conditions** | 1. The doctor can’t exist |
| **Post-conditions** | 1. The doctor exists |
| **Steps** | 1. Enter doctor name<br>2. Enter doctor age<br>3. Enter doctor specialty |
| **Exceptions** | (unspecified) |

| **Scenario** | **Manage Doctors** |
|---|---|
| **Description** | Remove a doctor from the system |
| **Pre-conditions** | 1. The doctor has to exist on the system |
| **Post-conditions** | 1. The doctor does no exist on the system<br>2. The doctor is not associated to any hospital<br>3. Does not exist any appointment for that doctor |
| **Steps** | 1. Choose a doctor to be removed |
| **Exceptions** | (unspecified) |

| **Scenario** | **Manage Doctors** |
|---|---|
| **Description** | List the all doctors |
| **Pre-conditions** | N/A |
| **Post-conditions** | N/A |
| **Steps** | N/A |
| **Exceptions** | (unspecified) |

| **Scenario** | **Manage Doctors** |
|---|---|
| **Description** | List the all doctors filtered by field |
| **Pre-conditions** | N/A |
| **Post-conditions** | N/A |
| **Steps** | 1. Enter the value of the field |
| **Exceptions** | (unspecified) |

| **Scenario** | **Manage patients** |
|---|---|
| **Description** | Add a patient on the system |
| **Pre-conditions** | 1. The patient can't exist |
| **Post-conditions** | 1. The patient exists |
| **Steps** | 1. Enter patient name<br>2. Enter patient age<br>3. Enter patient observation |
| **Exceptions** | (unspecified) |

| **Scenario** | **Manage patients** |
|---|---|
| **Description** | Remove a patient from the system |
| **Pre-conditions** | 1. The patient has to exist on the system |
| **Post-conditions** | 1. The patient does no exist on the system<br>2. Does not exist any appointment for that patient |
| **Steps** | 1. Choose a patient to be removed |
| **Exceptions** | (unspecified) |

| **Scenario** | **Manage patients** |
|---|---|
| **Description** | Add clinical observations to a patient |
| **Pre-conditions** | 1. The patient has to exist |
| **Post-conditions** | 1. The patient has to exist<br>2. The number of clinical observations for the patient increased |
| **Steps** | 1. Choose a patient<br>2. Introduce an observation about the patient |
| **Exceptions** | (unspecified) |

| **Scenario** | **Manage patients** |
|---|---|
| **Description** | See all patients |
| **Pre-conditions** | N/A |
| **Post-conditions** | N/A |
| **Steps** | N/A |
| **Exceptions** | (unspecified) |

| **Scenario** | **Manage appointments** |
|---|---|
| **Description** | Make an appointment associated to a doctor at certain time |
| **Pre-conditions** | 1. The hospital where the appointment will take place has to exist<br>2. The doctor, that will see the patient has to exist<br>3. The patient that goes to the appointment has to exist<br>4. The doctor has to be associated with the patient<br>5. The doctor and patient can not have overlapped appointments<br>6. The appointment can not already  exist |
| **Post-conditions** | 1. The hospital where the appointment will take place has to exist<br>2. The doctor, that will see the patient has to exist<br>3. The patient that goes to the appointment has to exist<br>4. The doctor has to be associated with the patient<br>5. The doctor and patient can not have overlapped appointments<br>6. The appointment  exist |
| **Steps** | 1. Choose a hospital<br>2. Choose a doctor<br>3. Choose a patient<br>4. Choose the year when the appointment will take place<br>5. Choose the month when the appointment will take place<br>6. Choose the day when the appointment will take place<br>7. Choose the hour when the appointment will take place<br>8. Choose the minute when the appointment will take place |
| **Exceptions** | (unspecified) |

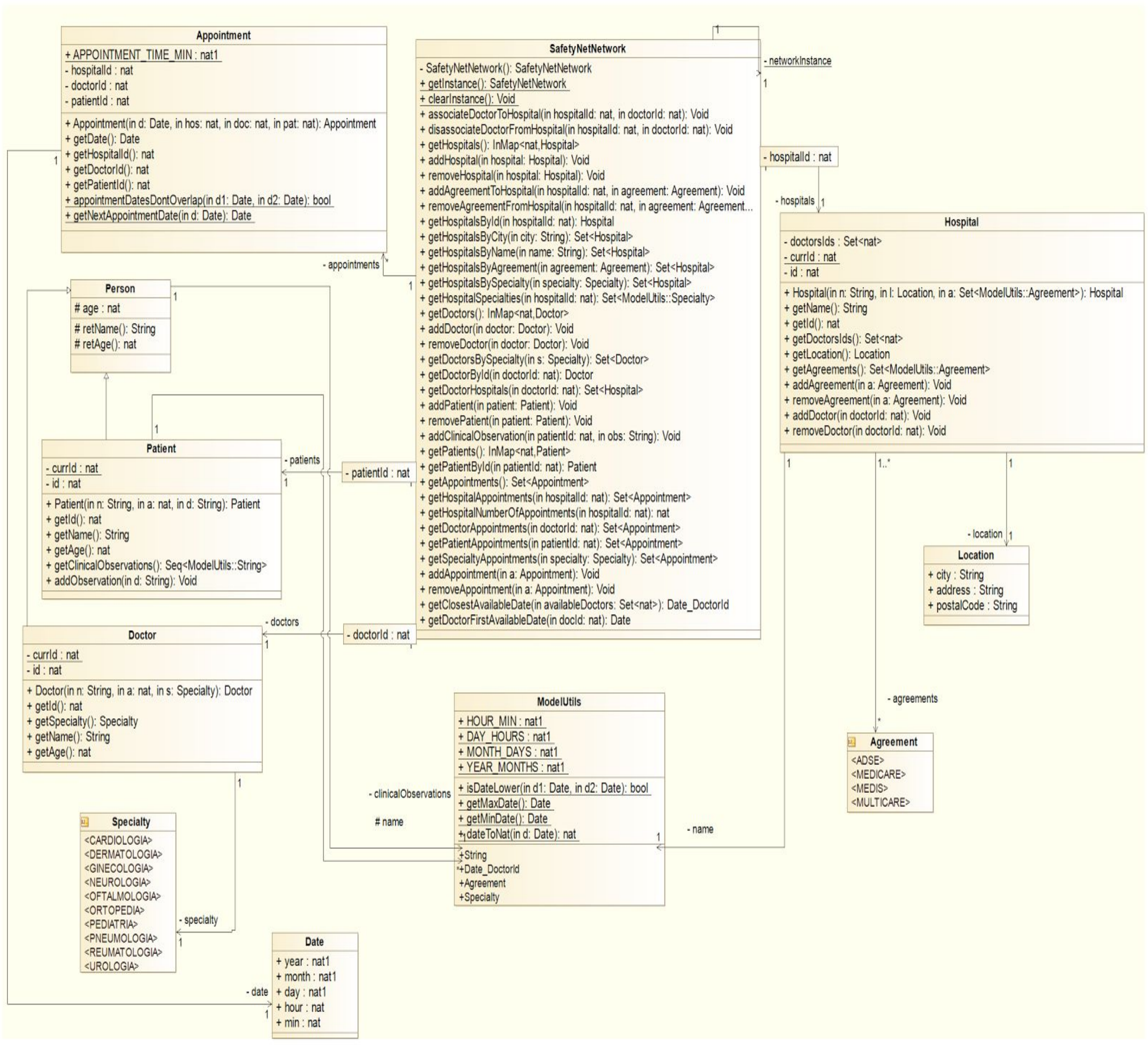
| **Scenario** | **Manage appointments** |
|---|---|
| **Description** | Cancel an appointment associated to a doctor at certain time |
| **Pre-conditions** | 1. The hospital where the appointment will take place has to exist<br>2. The doctor, that will see the patient has to exist<br>3. The patient that goes to the appointment has to exist<br>4. The doctor has to be associated with the patient<br>5. The appointment has to exist |
| **Post-conditions** | 1. The hospital where the appointment will take place has to exist<br>2. The doctor, that will see the patient has to exist<br>3. The patient that goes to the appointment has to exist<br>4. The doctor has to be associated with the patient<br>5. The appointment can not exist |
| **Steps** | 1. Select the appointment |
| **Exceptions** | (unspecified) |

| **Scenario** | **Manage appointments** |
|---|---|
| **Description** | See all appointments |
| **Pre-conditions** | N/A |
| **Post-conditions** | N/A |
| **Steps** | N/A |
| **Exceptions** | (unspecified) |

| **Scenario** | **Manage appointments** |
|---|---|
| **Description** | See all appointments filtered by field |
| **Pre-conditions** | N/A |
| **Post-conditions** | N/A |
| **Steps** | 1. Enter the value of the field |
| **Exceptions** | (unspecified) |

| **Scenario** | **Manage appointments** |
|---|---|
| **Description** | Get first available date for an appointment |
| **Pre-conditions** | 1. All the available doctors are registered on the network |
| **Post-conditions** | 1. The doctor that have the closest available date is registered on the network |
| **Steps** | 1. Select the specialty<br>2. Choose or don’t the hospital<br>3. Choose or don’t the doctor |
| **Exceptions** | (unspecified) |

## 2.3. Class model


Appointment
+ APPOINTMENT_TIME_MIN : nat1
- hospitalId : nat
- doctorId : nat
- patientId : nat
+ Appointment(in d: Date, in hos: nat, in doc: nat, in pat: nat): Appointment
+ getDate(): Date
+ getHospitalId(): nat
+ getDoctorId(): nat
+ getPatientId(): nat
+ appointmentDatesDontOverlap(in d1: Date, in d2: Date): bool
+ getNextAppointmentDate(in d: Date): Date
SafetyNetNetwork
- SafetyNetNetwork(): SafetyNetNetwork
+ getInstance(): SafetyNetNetwork
+ clearInstance(): Void
+ associateDoctorToHospital(in hospitalId: nat, in doctorId: nat): Void
+ disassociateDoctorFromHospital(in hospitalId: nat, in doctorId: nat): Void
+ getHospitals(): InMap<nat,Hospital>
+ addHospital(in hospital: Hospital): Void
+ removeHospital(in hospital: Hospital): Void
+ addAgreementToHospital(in hospitalId: nat, in agreement: Agreement): Void
+ removeAgreementFromHospital(in hospitalId: nat, in agreement: Agreement...
+ getHospitalsById(in hospitalId: nat): Hospital
+ getHospitalsByCity(in city: String): Set<Hospital>
+ getHospitalsByName(in name: String): Set<Hospital>
+ getHospitalsByAgreement(in agreement: Agreement): Set<Hospital>
+ getHospitalsBySpecialty(in specialty: Specialty): Set<Hospital>
+ getHospitalSpecialties(in hospitalId: nat): Set<ModelUtils::Specialty>
+ getDoctors(): InMap<nat,Doctor>
+ addDoctor(in doctor: Doctor): Void
+ removeDoctor(in doctor: Doctor): Void
+ getDoctorsBySpecialty(in s: Specialty): Set<Doctor>
+ getDoctorById(in doctorId: nat): Doctor
+ getDoctorHospitals(in doctorId: nat): Set<Hospital>
+ addPatient(in patient: Patient): Void
+ removePatient(in patient: Patient): Void
+ addClinicalObservation(in patientId: nat, in obs: String): Void
+ getPatients(): InMap<nat,Patient>
+ getPatientById(in patientId: nat): Patient
+ getAppointments(): Set<Appointment>
+ getHospitalAppointments(in hospitalId: nat): Set<Appointment>
+ getHospitalNumberOfAppointments(in hospitalId: nat): nat
+ getDoctorAppointments(in doctorId: nat): Set<Appointment>
+ getPatientAppointments(in patientId: nat): Set<Appointment>
+ getSpecialtyAppointments(in specialty: Specialty): Set<Appointment>
+ addAppointment(in a: Appointment): Void
+ removeAppointment(in a: Appointment): Void
+ getClosestAvailableDate(in availableDoctors: Set<nat>): Date_DoctorId
+ getDoctorFirstAvailableDate(in docId: nat): Date
1
- networkInstance
1
- hospitalId : nat
- hospitals
1
Hospital
- doctorsIds : Set<nat>
- currId : nat
- id : nat
+ Hospital(in n: String, in l: Location, in a: Set<ModelUtils::Agreement>): Hospital
+ getName(): String
+ getId(): nat
+ getDoctorsIds(): Set<nat>
+ getLocation(): Location
+ getAgreements(): Set<ModelUtils::Agreement>
+ addAgreement(in a: Agreement): Void
+ removeAgreement(in a: Agreement): Void
+ addDoctor(in doctorId: nat): Void
+ removeDoctor(in doctorId: nat): Void
1
1..*
1
- location
1
Location
+ city : String
+ address : String
+ postalCode : String
- agreements
*
Agreement
<ADSE>
<MEDICARE>
<MEDIS>
<MULTICARE>
- appointments
*
1
1
Person
# age : nat
# retName(): String
# retAge(): nat
1
1
Patient
- currId : nat
- id : nat
+ Patient(in n: String, in a: nat, in d: String): Patient
+ getId(): nat
+ getName(): String
+ getAge(): nat
+ getClinicalObservations(): Seq<ModelUtils::String>
+ addObservation(in d: String): Void
- patients
1
- patientId : nat
- doctors
1
- doctorId : nat
Doctor
- currId : nat
- id : nat
+ Doctor(in n: String, in a: nat, in s: Specialty): Doctor
+ getId(): nat
+ getSpecialty(): Specialty
+ getName(): String
+ getAge(): nat
1
ModelUtils
+ HOUR_MIN : nat1
+ DAY_HOURS : nat1
+ MONTH_DAYS : nat1
+ YEAR_MONTHS : nat1
+ isDateLower(in d1: Date, in d2: Date): bool
+ getMaxDate(): Date
+ getMinDate(): Date
+ dateToNat(in d: Date): nat
+String
+Date_DoctorId
+Agreement
+Specialty
- clinicalObservations
# name
1
- name
Specialty
<CARDIOLOGIA>
<DERMATOLOGIA>
<GINECOLOGIA>
<NEUROLOGIA>
<OFTALMOLOGIA>
<ORTOPEDIA>
<PEDIATRIA>
<PNEUMOLOGIA>
<REUMATOLOGIA>
<UROLOGIA>
- specialty
1
Date
+ year : nat1
+ month : nat1
+ day : nat1
+ hour : nat
+ min : nat
- date
1

| Class | Description |
|---|---|
| Person | Superclass, contains the information related to a person. |
| Doctor | SubClass of Person, implements a doctor. |
| Patient | SubClass of Person, implements a patient. |
| Appointment | Implements all the appointments details and necessary methods to manage them. |
| Hospital | Defines and manages a hospital |
| SafetyNetNetwork | Manages all the information of the system, managing all the doctors, appointments, patients and hospitals, becoming the core model of the system. |
| ModelUtils | Implements useful types and functions, that the majority of the previous classes use. |
| MyTestCase | Superclass of SystemTest, implements assertEqual, assertTrue and assertFalse. |
| SystemTest | Defines the test/usage scenarios and test cases for our Safety Net Hospital model. |

# 3. Formal VDM++ model

## 3.1. Class Person

```
class Person
instance variables
  protected name: ModelUtils`String := [];
  protected age: nat;

operations

         protected retName: () ==> ModelUtils`String
         retName() == (
                        return name;
         );

         protected retAge: () ==> nat
         retAge() == (
                        return age;
         );


end Person
```

| Function or operation | Line | Coverage | Calls |
|---|---|---|---|
| retAge | 13 | 100.0% | 48 |
| retName | 8 | 100.0% | 48 |
| Person.vdmpp | | 100.0% | 96 |

## 3.2. Class Doctor

```
class Doctor is subclass of Person
instance variables
        private specialty: ModelUtils`Specialty;
        private static currId : nat := 0;
        private id : nat := currId;

        inv age > 18;
operations
        --contructor
                public Doctor : ModelUtils`String * nat * ModelUtils`Specialty ==> Doctor
                Doctor(n, a, s) == (
                atomic (
                        name := n;
                        age := a;
                        specialty := s;
                        currId := currId + 1;
                );
                return self;
                );

                --get doctor id
                public pure getId: () ==> nat
                getId() == (
                return id
                );


                --get doctor specialties
                public pure getSpecialty: () ==> ModelUtils`Specialty
                getSpecialty() == (
                return specialty
                );

         --   getName
         public getName: () ==> ModelUtils`String
         getName() == (
                        return retName();
         );

         --   getAge
         public getAge: () ==> nat
         getAge() == (
                        return retAge();
         );

end Doctor
```

| Function or operation | Line | Coverage | Calls |
|---|---|---|---|
| Doctor | 10 | 100.0% | 402 |
| getAge | 41 | 100.0% | 24 |
| getId | 22 | 100.0% | 2925 |
| getName | 35 | 100.0% | 24 |
| getSpecialty | 29 | 100.0% | 344 |
| Doctor.vdmpp |  | 100.0% | 3719 |

## 3.3. Class Patient

```
class Patient is subclass of Person
instance variables
        private clinicalObservations: seq of ModelUtils`String := [];
        private static currId : nat := 0;
        private id : nat := currId;

operations
        public Patient : ModelUtils`String * nat * ModelUtils`String ==> Patient
        Patient(n, a, d) == (
                atomic (
                name  := n;
                age:= a;
                clinicalObservations:= clinicalObservations ^ [d];
                currId := currId + 1;
                );
                return self;
        );

        --get patient id
        public pure getId: () ==> nat
        getId() == (
                return id
        );

        --      getName
         public getName: () ==> ModelUtils`String
         getName() == (
                        return retName();
         );

         --   getAge
         public getAge: () ==> nat
         getAge() == (
                        return retAge();
         );

        --get clinical observations
        public pure getClinicalObservations: () ==> seq of ModelUtils`String
        getClinicalObservations() == (
                return clinicalObservations
        );


        --add clinical observation
        public addObservation: ModelUtils`String ==> ()
        addObservation(d) == (
                clinicalObservations := clinicalObservations ^ [d];
        )
        post len clinicalObservations = len clinicalObservations~ + 1 and
                 exists i in set inds clinicalObservations & clinicalObservations(i) = d;


end Patient
```

| Function or operation | Line | Coverage | Calls |
|---|---|---|---|
| Patient | 8 | 100.0% | 164 |
| addObservation | 45 | 100.0% | 24 |
| getAge | 32 | 100.0% | 24 |
| getClinicalObservations | 38 | 100.0% | 48 |
| getId | 20 | 100.0% | 1200 |
| getName | 26 | 100.0% | 24 |
| Patient.vdmpp | | 100.0% | 1484 |

## 3.4. Class Appointment

```
class Appointment
values
 public APPOINTMENT_TIME_MIN = 30; -- duration of an appointment in minutes
instance variables
        private date : ModelUtils`Date;
  private hospitalId: nat;
  private doctorId: nat;
  private patientId: nat;

  -- inv
        inv hospitalId in set dom SafetyNetNetwork`getInstance().getHospitals() and   -- |
                        doctorId in set dom SafetyNetNetwork`getInstance().getDoctors() and       --
|Check if they are in the network
                        patientId in set dom SafetyNetNetwork`getInstance().getPatients() and     -- |
                        doctorId in set
SafetyNetNetwork`getInstance().getHospitals()(hospitalId).getDoctorsIds(); --check if the doctor works
in that hospital

operations
        --constructor
        public Appointment: ModelUtils`Date * nat * nat * nat ==> Appointment
        Appointment(d,hos, doc, pat) == (
                atomic (
                date := d;
                hospitalId := hos;
                doctorId := doc;
                patientId:= pat;
                );
                return self;
        );

                --get appointment date
                public pure getDate: () ==> ModelUtils`Date
                getDate() == (
                return date
                );

                --get appointment hospital
                public pure getHospitalId: () ==> nat
                getHospitalId() == (
                return hospitalId
                );

                --get appointment doctor
                public pure getDoctorId: () ==> nat
                getDoctorId() == (
                return doctorId
                );

                --get appointment patient
                public pure getPatientId: () ==> nat
                getPatientId() == (
                return patientId
                );

functions
        --checks if two dates are equal
        static public appointmentDatesDontOverlap: ModelUtils`Date * ModelUtils`Date -> bool
        appointmentDatesDontOverlap(d1,d2) == (
                (ModelUtils`dateToNat(getNextAppointmentDate(d1)) <= ModelUtils`dateToNat(d2)) or
                (ModelUtils`dateToNat(getNextAppointmentDate(d2)) <= ModelUtils`dateToNat(d1))
        );

        --get next appointment slot date
        static public getNextAppointmentDate: ModelUtils`Date -> ModelUtils`Date
        getNextAppointmentDate(d) == (

                if(d.month = 12 and d.day=30 and d.hour = 23 and d.min >= (ModelUtils`HOUR_MIN -
APPOINTMENT_TIME_MIN)) then
                mk_ModelUtils`Date(d.year + 1, 1, 1, 00, (d.min + APPOINTMENT_TIME_MIN) mod
ModelUtils`HOUR_MIN)
                elseif(d.day=30 and d.hour = 23 and d.min >= (ModelUtils`HOUR_MIN -
APPOINTMENT_TIME_MIN)) then
                mk_ModelUtils`Date(d.year, d.month + 1, 1, 00, (d.min + APPOINTMENT_TIME_MIN) mod
ModelUtils`HOUR_MIN)
                elseif (d.hour = 23 and d.min >= (ModelUtils`HOUR_MIN - APPOINTMENT_TIME_MIN)) then
                mk_ModelUtils`Date(d.year, d.month, d.day+1, 00, (d.min + APPOINTMENT_TIME_MIN) mod
ModelUtils`HOUR_MIN)
                elseif (d.min >= (ModelUtils`HOUR_MIN - APPOINTMENT_TIME_MIN)) then
                mk_ModelUtils`Date(d.year, d.month, d.day, d.hour + 1, (d.min + APPOINTMENT_TIME_MIN)
mod ModelUtils`HOUR_MIN)
                else
                mk_ModelUtils`Date(d.year, d.month, d.day, d.hour, d.min + APPOINTMENT_TIME_MIN)
        );


end Appointment
```

| Function or operation | Line | Coverage | Calls |
|---|---|---|---|
| Appointment | 18 | 100.0% | 172 |
| appointmentDatesDontOverlap | 55 | 100.0% | 324 |
| getDate | 30 | 100.0% | 833 |
| getDoctorId | 42 | 100.0% | 1285 |
| getHospitalId | 36 | 100.0% | 264 |
| getNextAppointmentDate | 62 | 100.0% | 22 |
| getPatientId | 48 | 100.0% | 919 |
| Appointment.vdmpp |  | 100.0% | 3819 |

## 3.5. Class Hospital

```
class Hospital
instance variables
        private name: ModelUtils`String := [];
        private location: ModelUtils`Location;
        private agreements: set of ModelUtils`Agreement := {};
        private doctorsIds: set of nat := {};
        private static currId : nat := 0;
        private id : nat := currId;

        inv forall d in set doctorsIds & d in set dom SafetyNetNetwork`getInstance().getDoctors();

operations

        --constructor
        public Hospital: ModelUtils`String * ModelUtils`Location * set of ModelUtils`Agreement ==>
Hospital
        Hospital(n,l,a) == (
                atomic (
                name := n;
                location := l;
                agreements := a;
                currId := currId + 1;
                );
                return self
        );

        --get hospital name
        public pure getName: () ==> ModelUtils`String
        getName() == (
                return name
        );

        --get hospital id
        public pure getId: () ==> nat
        getId() == (
                return id
        );

        --get hospital doctors
        public pure getDoctorsIds: () ==> set of nat
        getDoctorsIds() == (
                return doctorsIds
        );

        --get hospital location
        public pure getLocation: () ==> ModelUtils`Location
        getLocation() == (
                return location
        );

        --get hospital agreements
        public pure getAgreements: () ==> set of ModelUtils`Agreement
        getAgreements() == (
                return agreements
        );

        --add agreement
        public addAgreement: ModelUtils`Agreement ==> ()
        addAgreement(a) == (
                agreements := agreements union {a};
        )
        pre a not in set agreements
        post a in set agreements;


        --remove agreement
        public removeAgreement: ModelUtils`Agreement ==> ()
        removeAgreement(a) == (
                agreements := agreements \ {a};
        )
        pre a in set agreements
        post a not in set agreements;

        --add doctor
        public addDoctor: nat ==> ()
        addDoctor(doctorId) == (
                doctorsIds := doctorsIds union {doctorId}
        )
        pre doctorId not in set doctorsIds
        post doctorId in set doctorsIds;


        --remove doctor
        public removeDoctor: nat ==> ()
        removeDoctor(doctorId) == (
                doctorsIds := doctorsIds \ {doctorId}
        )
        pre doctorId in set doctorsIds
        post doctorId not in set doctorsIds;

end Hospital
```

| Function or operation | Line | Coverage | Calls |
|---|---|---|---|
| Hospital | 15 | 100.0% | 476 |
| addAgreement | 57 | 100.0% | 9 |
| addDoctor | 72 | 100.0% | 318 |
| getAgreements | 51 | 100.0% | 171 |
| getDoctorsIds | 39 | 100.0% | 1120 |
| getId | 33 | 100.0% | 2844 |
| getLocation | 45 | 100.0% | 72 |
| getName | 27 | 100.0% | 27292 |
| removeAgreement | 65 | 100.0% | 9 |
| removeDoctor | 81 | 100.0% | 36 |
| Hospital.vdmpp | | 100.0% | 32347 |

## 3.6. Class SafetyNetNetwork

```
class SafetyNetNetwork
instance variables
                private hospitals: inmap nat to Hospital := { |-> };
                private doctors: inmap nat to Doctor := { |-> };
                private patients: inmap nat to Patient := { |-> };
                private appointments: set of Appointment := {};
                private static networkInstance: SafetyNetNetwork := new SafetyNetNetwork();

                inv not exists h1, h2 in set rng hospitals &
                h1 <> h2 and h1.getName() = h2.getName();

operations

                --constructor
                private SafetyNetNetwork: () ==> SafetyNetNetwork
                SafetyNetNetwork() == return self
                post hospitals = { |-> } and  doctors = { |-> };

                --get network instance
                public pure static getInstance: () ==> SafetyNetNetwork
                getInstance() == return networkInstance
                post isofclass(SafetyNetNetwork,RESULT);


                --clear network instance
                public static clearInstance: () ==> ()
                clearInstance() == (
                networkInstance := new SafetyNetNetwork();
                );

                --associate doctor to hospital
                public associateDoctorToHospital : nat * nat ==> ()
                associateDoctorToHospital(hospitalId, doctorId) == (
                        hospitals(hospitalId).addDoctor(doctorId)
                )
                pre hospitalId in set dom hospitals and doctorId in set dom doctors and doctorId not
in set hospitals(hospitalId).getDoctorsIds()
          post hospitalId in set dom hospitals and doctorId in set dom doctors and doctorId in set
hospitals(hospitalId).getDoctorsIds();


                --disassociate doctor from hospital
                public disassociateDoctorFromHospital : nat * nat ==> ()
                disassociateDoctorFromHospital(hospitalId, doctorId) == (
                        hospitals(hospitalId).removeDoctor(doctorId)
                )
                pre hospitalId in set dom hospitals and doctorId in set dom doctors and doctorId in
set hospitals(hospitalId).getDoctorsIds()
                post hospitalId in set dom hospitals and doctorId in set dom doctors and doctorId not
in set hospitals(hospitalId).getDoctorsIds();

                ----------------------------------------------------------
                -----------------Hospital---------------------------------
                ----------------------------------------------------------
                -- get hospitals
                public pure getHospitals : () ==> inmap nat to Hospital
                getHospitals() == (
                return hospitals;
```

```
        );

        --add hospital
        public addHospital: Hospital ==> ()
        addHospital(hospital) == (
        hospitals := hospitals ++  { hospital.getId() |-> hospital};
        )
        pre {hospital.getId() } <: hospitals = { |-> }
        post {hospital.getId() } <: hospitals = { hospital.getId() |-> hospital } ;

        --remove an hospital
        public removeHospital: Hospital ==> ()
        removeHospital(hospital) == (
        hospitals := {hospital.getId()} <-: hospitals;
        --cancel appointments in that hospital
        for all a in set appointments do
                if(a.getHospitalId() = hospital.getId()) then
                        removeAppointment(a);
        )
  pre {hospital.getId()} <: hospitals = { hospital.getId() |-> hospital }
        post {hospital.getId()} <: hospitals = { |-> } and
         forall a in set appointments & a.getHospitalId() <> hospital.getId();

        -- add agreement to hospital
        public addAgreementToHospital: nat * ModelUtils`Agreement ==> ()
        addAgreementToHospital(hospitalId, agreement) == (
                hospitals(hospitalId).addAgreement(agreement);
        )
        pre {hospitalId} <: hospitals = { hospitalId |-> hospitals(hospitalId) };

        -- remove agreement from hospital
        public removeAgreementFromHospital: nat * ModelUtils`Agreement ==> ()
        removeAgreementFromHospital(hospitalId, agreement) == (
                hospitals(hospitalId).removeAgreement(agreement);
        )
        pre {hospitalId} <: hospitals = { hospitalId |-> hospitals(hospitalId) };

        -------search hospitals------------------------------------

        ----------get hospitals by id
        public pure getHospitalsById: nat ==> Hospital
        getHospitalsById(hospitalId) == (
                return hospitals(hospitalId);
        )
        pre hospitalId in set dom hospitals
        post RESULT.getId() = hospitalId;

        ----------get hospitals by city
        public pure getHospitalsByCity: ModelUtils`String ==> set of Hospital
        getHospitalsByCity(city) == (
                dcl res : set of Hospital := {};
                for all h in set rng hospitals do
                                if(h.getLocation().city = city) then
                                        res := res union {h};
                return res
        );
        ----------get hospitals by name
        public pure getHospitalsByName: ModelUtils`String ==> set of Hospital
        getHospitalsByName(name) == (
                dcl res : set of Hospital := {};
                for all h in set rng hospitals do
                                if(h.getName() = name) then
```

```
                                        res := res union {h};
        return res
);

----------get hospitals by agreement
public pure getHospitalsByAgreement: ModelUtils`Agreement ==> set of Hospital
getHospitalsByAgreement(agreement) == (
        dcl res : set of Hospital := {};
        for all h in set rng hospitals do
                        if(agreement in set h.getAgreements()) then
                                res := res union {h};
        return res
);

----------get hospitals by specialty
public pure getHospitalsBySpecialty: ModelUtils`Specialty ==> set of Hospital
getHospitalsBySpecialty(specialty) == (
        dcl res : set of Hospital := {};
        for all h in set rng hospitals do
                for all d in set h.getDoctorsIds() do
                        if(specialty = doctors(d).getSpecialty()) then
                                res := res union {h};
        return res
);

-------------------End hospital search--------------------------------

-- get hospitals specialties
public pure getHospitalSpecialties: nat ==> set of ModelUtils`Specialty
getHospitalSpecialties(hospitalId) == (
        dcl res : set of ModelUtils`Specialty := {};
                for all doctorId in set hospitals(hospitalId).getDoctorsIds() do
                                res := res union {doctors(doctorId).getSpecialty()};
        return res
);

--------------------------------------------------------------
-----------------------end hospital --------------------------
--------------------------------------------------------------

--------------------------------------------------------
-----------------------Doctors -------------------------
--------------------------------------------------------
-- get doctors
public pure getDoctors : () ==> inmap nat to Doctor
getDoctors() == (
return doctors;
);

--add doctor
public addDoctor: Doctor ==> ()
addDoctor(doctor) == (
doctors := doctors ++  { doctor.getId() |-> doctor};
)
pre {doctor.getId() } <: doctors = { |-> }
post {doctor.getId() } <: doctors = { doctor.getId() |-> doctor };

--remove doctor from the network and from all the hospitals where he works
public removeDoctor: Doctor ==> ()
removeDoctor(doctor) == (
--remove doctor from network
doctors := {doctor.getId()} <-: doctors;
```

```
        --remove doctor from hospitals where he works
        for all h in set rng hospitals do
                if(doctor.getId() in set h.getDoctorsIds()) then
                        h.removeDoctor(doctor.getId());
        --cancel doctor appointments
        for all a in set appointments do
                if(a.getDoctorId() = doctor.getId()) then
                        removeAppointment(a);
        )
      pre {doctor.getId()} <: doctors = { doctor.getId() |-> doctor }
        post {doctor.getId()} <: doctors = { |-> } and
                 forall h in set rng hospitals & doctor.getId() not in set h.getDoctorsIds()
and

                 forall a in set appointments & a.getDoctorId() <> doctor.getId();

        --search doctors--------------------------------
        ----------get doctor by specialty
        public pure getDoctorsBySpecialty: ModelUtils`Specialty ==> set of Doctor
        getDoctorsBySpecialty(s) == (
                dcl res : set of Doctor := {};
                for all d in set rng doctors do
                                if(d.getSpecialty() = s) then
                                        res := res union {d};
                return res
        );

        ----------get doctor by id
        public pure getDoctorById: nat ==> Doctor
        getDoctorById(doctorId) == (
                return doctors(doctorId);
        )
        pre doctorId in set dom doctors
        post RESULT.getId() = doctorId;

        ----------get hospitals where a doctor works
        public pure getDoctorHospitals: nat ==> set of Hospital
        getDoctorHospitals(doctorId) == (
                dcl res : set of Hospital := {};
                for all h in set rng hospitals do
                        if(doctorId in set h.getDoctorsIds()) then
                                res := res union {h};
                return res
        );

        ---------------------------------------------------------
        -----------------------End Doctors ----------------------
        ---------------------------------------------------------


        -----------------------------------------------------------
        -----------------Patients----------------------------------
        -----------------------------------------------------------

        --add patient
        public addPatient: Patient ==> ()
        addPatient(patient) == (
        patients := patients ++  { patient.getId() |-> patient};
        )
        pre {patient.getId() } <: patients = { |-> }
        post {patient.getId() } <: patients = { patient.getId() |-> patient };

        --remove patient
        public removePatient: Patient ==> ()
```

```
                removePatient(patient) == (
                --remove patient appointments
                for all a in set appointments do
                        if(a.getPatientId() = patient.getId()) then
                                removeAppointment(a);
                --remove patient from network
                patients := {patient.getId()} <-: patients;
                )
          pre {patient.getId()} <: patients = { patient.getId() |-> patient }
                post {patient.getId()} <: patients = { |-> } and forall a in set appointments &
a.getPatientId() <> patient.getId();

                --add clinical observation
                public addClinicalObservation: nat * ModelUtils`String ==> ()
                addClinicalObservation(patientId, obs) == (
                patients(patientId).addObservation(obs);
                )
          pre {patientId} <: patients = { patientId |-> patients(patientId) } ;

          -- get patients
                public pure getPatients : () ==> inmap nat to Patient
                getPatients() == (
                return patients;
                );

                ----------get patient by id
                public pure getPatientById: nat ==> Patient
                getPatientById(patientId) == (
                        return patients(patientId);
                )
                pre patientId in set dom patients
                post RESULT.getId() = patientId;

                ---------------------------------------------------------
                ------------------End Patients----------------------------
                -----------------------------------------------------------

                ---------------------------------------------------------
                ----------------------Appointments----------------------
                ---------------------------------------------------------

                -- get appointments
                public pure getAppointments : () ==> set of Appointment
                getAppointments() == (
                return appointments;
                );

                --get hospital appointments
                public pure getHospitalAppointments: nat ==> set of Appointment
                getHospitalAppointments(hospitalId) == (
                dcl res: set of Appointment := {};
                for all a in set appointments do
                        if(a.getHospitalId() = hospitalId) then
                                        res := res union {a};

                return res
                )
                pre {hospitalId} <: hospitals = { hospitalId |-> hospitals(hospitalId) }
                post forall a in set RESULT & isofclass(Appointment,a) and a.getHospitalId() =
hospitalId;

                -- total number of appointments in a hospital
                public pure getHospitalNumberOfAppointments: nat ==> nat
                getHospitalNumberOfAppointments(hospitalId) == (
                        return card getHospitalAppointments(hospitalId);
                )
                pre hospitalId in set dom hospitals;

                --get doctor appointments
                public pure getDoctorAppointments: nat ==> set of Appointment
                getDoctorAppointments(doctorId) == (
                dcl res: set of Appointment := {};
                for all a in set appointments do
                        if(a.getDoctorId() = doctorId) then
                                res := res union {a};
                return res
                )
                pre {doctorId} <: doctors = { doctorId |-> doctors(doctorId) }
                post forall a in set RESULT & isofclass(Appointment,a) and a.getDoctorId() = doctorId;

                --get patient appointments
                public pure getPatientAppointments: nat ==> set of Appointment
                getPatientAppointments(patientId) == (
                dcl res: set of Appointment := {};
                for all a in set appointments do
                        if(a.getPatientId() = patientId) then
                                res := res union {a};
                return res
                )
                pre {patientId} <: patients = { patientId |-> patients(patientId) }
                post forall a in set RESULT & isofclass(Appointment,a) and a.getPatientId() =
patientId;

                --get specialty appointments
                public pure getSpecialtyAppointments: ModelUtils`Specialty ==> set of Appointment
                getSpecialtyAppointments(specialty) == (
                dcl res: set of Appointment := {};
                for all a in set appointments do
                        if(doctors(a.getDoctorId()).getSpecialty() = specialty) then
                                res := res union {a};
                return res
                )
                post forall a in set RESULT & isofclass(Appointment,a) and
doctors(a.getDoctorId()).getSpecialty() = specialty;

                --add an Appointment
                public addAppointment: Appointment ==> ()
                addAppointment(a) == (
                appointments := appointments union {a};
                )
                pre forall ap in set getDoctorAppointments(a.getDoctorId()) union
getPatientAppointments(a.getPatientId()) & Appointment`appointmentDatesDontOverlap(ap.getDate(),
a.getDate())
                post a in set appointments;

                --remove an Appointment
                public removeAppointment: Appointment ==> ()
                removeAppointment(a) == (
                appointments := appointments \ {a};
                )
                pre a in set appointments
                post a not in set appointments;

                -- get closest appointment date available given a set of doctors
                -- example: pass the doctor ids with who you want to get an appointment and receive
the closest appointment date and the doctor available on that date
                public getClosestAvailableDate: set of nat ==> ModelUtils`Date_DoctorId
                 getClosestAvailableDate(availableDoctors) == (
                dcl minDate: ModelUtils`Date := ModelUtils`getMaxDate();
                dcl doctorId: nat;

                for all docId in set availableDoctors do
                (
                        dcl auxDate : ModelUtils`Date := getDoctorFirstAvailableDate(docId);
                        if(ModelUtils`isDateLower(auxDate, minDate)) then
                                                (
                                                doctorId := docId;
                                                minDate := auxDate;
                                                )
                );
                return mk_ModelUtils`Date_DoctorId(minDate, doctorId);
                )
                pre forall d in set availableDoctors & d in set dom doctors
                post RESULT.doctorId in set dom doctors;

                -- get the doctor first available date
                public getDoctorFirstAvailableDate: nat ==> ModelUtils`Date
                 getDoctorFirstAvailableDate(docId) == (
                dcl minDate: ModelUtils`Date := ModelUtils`getMaxDate();
                dcl occupiedDates: set of ModelUtils`Date := {};

                for all docAp in set getDoctorAppointments(docId) do
                        occupiedDates := occupiedDates union {docAp.getDate()};

                -- have to check if the first date is available
                occupiedDates := occupiedDates union {ModelUtils`getMinDate()};

                for all date in set occupiedDates do
                (
                        dcl auxDate : ModelUtils`Date := Appointment`getNextAppointmentDate(date);
                        if(ModelUtils`isDateLower(auxDate, minDate)) then
                                                if(forall docAp in set getDoctorAppointments(docId) &
Appointment`appointmentDatesDontOverlap(docAp.getDate(),auxDate)) then -- if the next appointment is
on a closer date than the actual minimum and one of the doctor has the date slot available, update the
minimum
                                                minDate := auxDate;
                );

                return minDate;
                )
                pre docId in set dom doctors;

                --------------------------------------------------------
                ----------------------End Appointments------------------
                --------------------------------------------------------

end SafetyNetNetwork
```

| Function or operation | Line | Coverage | Calls |
|---|---|---|---|
| SafetyNetNetwork | 15 | 100.0% | 606 |
| addAgreementToHospital | 80 | 100.0% | 13 |
| addAppointment | 353 | 100.0% | 448 |
| addClinicalObservation | 259 | 100.0% | 48 |
| addDoctor | 172 | 100.0% | 852 |
| addHospital | 59 | 100.0% | 971 |
| addPatient | 238 | 100.0% | 364 |
| associateDoctorToHospital | 32 | 100.0% | 694 |
| clearInstance | 26 | 100.0% | 581 |
| disassociateDoctorFromHospital | 41 | 100.0% | 25 |
| getAppointments | 289 | 100.0% | 240 |
| getClosestAvailableDate | 372 | 100.0% | 38 |
| getDoctorAppointments | 317 | 100.0% | 920 |
| getDoctorById | 211 | 100.0% | 72 |
| getDoctorFirstAvailableDate | 393 | 100.0% | 19 |
| getDoctorHospitals | 219 | 100.0% | 3 |
| getDoctors | 166 | 100.0% | 68 |
| getDoctorsBySpecialty | 201 | 100.0% | 3 |
| getHospitalAppointments | 295 | 100.0% | 5 |
| getHospitalClosestAvailableDate | 372 | 100.0% | 10 |
| getHospitalNumberOfAppointments | 308 | 100.0% | 8 |
| getHospitalSpecialties | 148 | 100.0% | 3 |
| getHospitals | 52 | 100.0% | 49 |
| getHospitalsByAgreement | 125 | 100.0% | 6 |
| getHospitalsByCity | 105 | 100.0% | 2 |
| getHospitalsById | 97 | 100.0% | 3 |
| getHospitalsByName | 115 | 100.0% | 3 |
| getHospitalsBySpecialty | 135 | 100.0% | 30 |
| getInstance | 20 | 100.0% | 253 |
| getPatientAppointments | 329 | 100.0% | 27 |
| getPatientById | 272 | 100.0% | 2 |
| getPatients | 266 | 100.0% | 17 |
| getSpecialtyAppointments | 341 | 100.0% | 3 |
| removeAgreementFromHospital | 88 | 100.0% | 1 |
| removeAppointment | 361 | 100.0% | 8 |
| removeDoctor | 180 | 100.0% | 2 |
| removeHospital | 67 | 100.0% | 3 |
| removePatient | 246 | 100.0% | 2 |
| SafetyNetNetwork.vdmpp |  | 100.0% | 795 |

## 3.7. Class ModelUtils

```
class ModelUtils
values
        public HOUR_MIN = 60; -- minutes in an hour
        public DAY_HOURS = 24; -- hour in a day
        public MONTH_DAYS = 30; -- days in a month
        public YEAR_MONTHS = 12; -- months in a year
types
        public String = seq of char;
        public Location :: city: String
                          address: String
                         postalCode: String;
        public Date :: year : nat1
                      month: nat1
                      day : nat1
                           hour : nat
                      min : nat
        inv d == d.year > 2017 and d.month <= YEAR_MONTHS and d.day <= MONTH_DAYS and d.hour <
DAY_HOURS and d.min < HOUR_MIN;
        public Date_DoctorId :: date : Date
                                                                                  doctorId: nat;

        public Agreement = <ADSE> | <MEDICARE> | <MEDIS> | <MULTICARE>;
        public Specialty = <ORTOPEDIA> | <CARDIOLOGIA> | <OFTALMOLOGIA> |
                               <DERMATOLOGIA> | <GINECOLOGIA> | <NEUROLOGIA> |
                          <PEDIATRIA> | <REUMATOLOGIA> | <UROLOGIA> |
                          <PNEUMOLOGIA>;

functions

        --checks if two dates are equal
        static public isDateLower: Date * Date -> bool
        isDateLower(d1,d2) == (
                dateToNat(d1) < dateToNat(d2)
        );

        --get max date
        static public getMaxDate: () -> Date
        getMaxDate() == (
                mk_ModelUtils`Date(99999,12,30,23,59)
        );

        --get min date
        static public getMinDate: () -> Date
        getMinDate() == (
                mk_ModelUtils`Date(2018,01,01,08,00)
        );

        --checks if two dates are equal
        static public dateToNat: Date -> nat
        dateToNat(d) == (
                d.year * 100000000 +
                d.month * 1000000 +
                d.day * 10000 +
                d.hour * 100 +
                d.min
        );

end ModelUtils
```

| Function or operation | Line | Coverage | Calls |
|---|---|---|---|
| dateToNat | 48 | 100.0% | 824 |
| getMaxDate | 36 | 100.0% | 11 |
| getMinDate | 42 | 100.0% | 11 |
| isDateLower | 30 | 100.0% | 88 |
| ModelUtils.vdmpp | | 100.0% | 934 |

# 4. Model validation

## 4.1. Class MyTestCase

```
class MyTestCase

operations


        protected assertTrue: bool ==> ()
        assertTrue(arg) ==
                return
        pre arg;

        protected assertFalse: bool ==> ()
        assertFalse(arg) ==
                return
        pre not arg;

        protected assertEqual: ? * ? ==> ()
        assertEqual(expected, actual) ==
                if expected <> actual then (
                IO`print("Actual value (");
                IO`print(actual);
                IO`print(") different from expected (");
                IO`print(expected);
                IO`println(")\n")
                )
        post expected = actual;

end MyTestCase
```

## 4.2. Class SystemTest

```
class SystemTest is subclass of MyTestCase
types
-- TODO Define types here
values
-- TODO Define values here
instance variables

                private safetyNet: SafetyNetNetwork := SafetyNetNetwork`getInstance();

operations


public static main: () ==> ()
main() == (

                dcl systemTest: SystemTest := new SystemTest();

                IO`println("network ");

                -- association hospital - doctor
                IO`print("test associateADoctorToAnHospital -> ");
                systemTest.testAssociateDoctorToAnHospital();
                IO`println("Success");

                IO`print("test disassociateADoctorToAnHospital -> ");
                systemTest.testDisassociateDoctorToAnHospital();
                IO`println("Success");


                -- Hospital
                IO`print("test addHospital -> ");
                systemTest.testAddHospital();
                IO`println("Success");

                IO`print("test removeHospital -> ");
                systemTest.testRemoveHospital();
                IO`println("Success");

                IO`print("test getAllHospitalsByLocation -> ");
                systemTest.testGetHospitalsByLocation();
                IO`println("Success");

                IO`print("test getAllHospitals -> ");
                systemTest.testGetAllHospitals();
                IO`println("Success");

                IO`print("test getHospitalsByName -> ");
                systemTest.testGetHospitalsByName();
                IO`println("Success");

                IO`print("test getHospitalsById -> ");
                systemTest.testGetHospitalsById();
                IO`println("Success");

                IO`print("test getHospitalsByAgreement -> ");
                systemTest.testGetHospitalsByAgreement();
                IO`println("Success");
```

```
IO`print("test getHospitalBySpecialtie -> ");
systemTest.testGetHospitalBySpecialtie();
IO`println("Success");

IO`print("test getHospitalSpecialties -> ");
systemTest.testGetHospitalSpecialties();
IO`println("Success");

-- Doctor
IO`print("test addDoctor -> ");
systemTest.testAddDoctor();
IO`println("Success");

IO`print("test getDoctors -> ");
systemTest.testGetAllDoctors();
IO`println("Success");

IO`print("test removeDoctor -> ");
systemTest.testRemoveDoctor();
IO`println("Success");

IO`print("test getDoctorHospitals -> ");
systemTest.testGetDoctorHospitals();
IO`println("Success");

IO`print("test getDoctorBySpecialtie-> ");
systemTest.testGetDoctorBySpecialtie();
IO`println("Success");

IO`print("test getDoctorById -> ");
systemTest.testGetDoctorById();
IO`println("Success");

-- Patient
IO`print("test addPatient -> ");
systemTest.testAddPatient();
IO`println("Success");

IO`print("test removePatient -> ");
systemTest.testRemovePatient();
IO`println("Success");

IO`print("test addObservation -> ");
systemTest.testAddObservation();
IO`println("Success");

IO`print("test getPatientById -> ");
systemTest.testGetPatientById();
IO`println("Success");


-- Appointement

IO`print("test addAppointment -> ");
systemTest.testAddAppointment();
IO`println("Success");

IO`print("test removeAppointment -> ");
systemTest.testRemoveAppointment();
IO`println("Success");

IO`print("test getSpecialtyAppointments -> ");
```

```
                systemTest.testGetSpecialtyAppointments();
                IO`println("Success");

                IO`print("test getHospitalClosestAvailableDate -> ");
                systemTest.testGetHospitalClosestAvailableDate();
                IO`println("Success");

                IO`print("test getNextAppointmentDate -> ");
                systemTest.testGetNextAppointmentDate();
                IO`println("Success");

                -- Agreement

                IO`print("test addAgreement-> ");
                systemTest.testAddAgreement();
                IO`println("Success");

                IO`print("test removeAgreement -> ");
                systemTest.testRemoveAgreement();
                IO`println("Success");

);

-- tests if a doctor is correctly added to the network
private testAddDoctor: () ==> ()
        testAddDoctor () == (

                dcl doc1: Doctor := new Doctor("jose",35,<ORTOPEDIA>);
                dcl doc2: Doctor := new Doctor("marcelo", 40 ,<CARDIOLOGIA>);

                safetyNet := SafetyNetNetwork`getInstance();

                safetyNet.addDoctor(doc1);
                safetyNet.addDoctor(doc2);

                assertEqual(doc1.getName(), "jose");
                assertEqual(doc2.getName(), "marcelo");

                assertEqual(doc1.getAge(), 35);
                assertEqual(doc2.getAge(), 40);

                assertEqual(doc1.getSpecialty(), <ORTOPEDIA>);
                assertEqual(doc2.getSpecialty(), <CARDIOLOGIA>);

                assertEqual( safetyNet.getDoctors(), {doc1.getId() |-> doc1, doc2.getId() |->doc2});

                safetyNet.clearInstance();

);

-- tests if the doctors are correctly obtained through the function getAllDoctors
private testGetAllDoctors: () ==> ()
        testGetAllDoctors () == (

                dcl doc1: Doctor := new Doctor("jose",35,<ORTOPEDIA>);
                dcl doc2: Doctor := new Doctor("marcelo", 40 ,<CARDIOLOGIA>);

                safetyNet := SafetyNetNetwork`getInstance();

                assertEqual(rng safetyNet.getDoctors(), {});

                safetyNet.addDoctor(doc1);
                safetyNet.addDoctor(doc2);

                assertEqual( safetyNet.getDoctors(), {doc1.getId() |-> doc1, doc2.getId() |->doc2});

                safetyNet.clearInstance();

);

-- verifies if a doctor is correctly removed from the network and associations with hospitals
private testRemoveDoctor: () ==> ()
        testRemoveDoctor () == (

                dcl doc1: Doctor := new Doctor("jose",35,<ORTOPEDIA>);
                dcl doc2: Doctor := new Doctor("marcelo", 40 ,<CARDIOLOGIA>);

                dcl hos1: Hospital := new Hospital("Hospital Sao Joao",mk_ModelUtils`Location("Porto",
"Alameda Prof. HernÃ¢ni Monteiro","4200-319"), {<ADSE>,<MEDICARE>});
                dcl hos2: Hospital := new Hospital("Hospital da Luz
Lisboa",mk_ModelUtils`Location("Lisboa", "Avenida LusÃada, nÂº 100","4700-959"), {<ADSE>,<MEDIS>,
<MULTICARE>});

                safetyNet := SafetyNetNetwork`getInstance();

                safetyNet.addHospital(hos1);
                safetyNet.addHospital(hos2);

                safetyNet.addDoctor(doc1);
                safetyNet.addDoctor(doc2);

                safetyNet.associateDoctorToHospital(hos1.getId(), doc1.getId());
                safetyNet.associateDoctorToHospital(hos2.getId(), doc2.getId());

                safetyNet.removeDoctor(doc2);

                -- check if was removed from the system
                assertEqual( safetyNet.getDoctors(), {doc1.getId() |-> doc1});

                -- check if the doctor was removed from all hospitals where he worked
                for all hs in set rng safetyNet.getHospitals() do
                assertFalse(doc2.getId() in set hs.getDoctorsIds());

                safetyNet.clearInstance();

);

-- verifies if a hospital is correctly added to the network
private testAddHospital: () ==> ()
        testAddHospital () == (

                dcl hos1: Hospital := new Hospital("Hospital Sao Joao",mk_ModelUtils`Location("Porto",
"Alameda Prof. HernÃ¢ni Monteiro","4200-319"), {<ADSE>,<MEDICARE>});
                dcl hos2: Hospital := new Hospital("Hospital da Luz
Lisboa",mk_ModelUtils`Location("Lisboa", "Avenida LusÃada, nÂº 100","4700-959"), {<ADSE>,<MEDIS>,
<MULTICARE>});

                safetyNet := SafetyNetNetwork`getInstance();

                safetyNet.addHospital(hos1);
                safetyNet.addHospital(hos2);

                assertEqual(safetyNet.getHospitals(), { hos1.getId() |-> hos1, hos2.getId() |->
hos2});
```

```
                safetyNet.clearInstance();

);

-- verifies if the hospital is correctly removed from the network and there is not any association
with a doctor
private testRemoveHospital: () ==> ()
        testRemoveHospital () == (

                dcl hos1: Hospital := new Hospital("Hospital Sao Joao",mk_ModelUtils`Location("Porto",
"Alameda Prof. HernÃ¢ni Monteiro","4200-319"), {<ADSE>,<MEDICARE>});
                dcl hos2: Hospital := new Hospital("Hospital da Luz
Lisboa",mk_ModelUtils`Location("Lisboa", "Avenida LusÃada, nÂº 100","4700-959"), {<ADSE>,<MEDIS>,
<MULTICARE>});

                safetyNet := SafetyNetNetwork`getInstance();

                safetyNet.addHospital(hos1);
                safetyNet.addHospital(hos2);

                safetyNet.removeHospital(hos1);

                assertEqual(safetyNet.getHospitals(), { hos2.getId() |-> hos2});

                safetyNet.clearInstance();

);

-- verifies if a doctor is correctly associated with a hospital
private testAssociateDoctorToAnHospital: () ==> ()
        testAssociateDoctorToAnHospital () == (

                dcl hos1: Hospital := new Hospital("Hospital Sao Joao",mk_ModelUtils`Location("Porto",
"Alameda Prof. HernÃ¢ni Monteiro","4200-319"), {<ADSE>,<MEDICARE>});
                dcl hos2: Hospital := new Hospital("Hospital da Luz
Lisboa",mk_ModelUtils`Location("Lisboa", "Avenida LusÃada, nÂº 100","4700-959"), {<ADSE>,<MEDIS>,
<MULTICARE>});

                dcl doc1: Doctor := new Doctor("jose",35,<ORTOPEDIA>);
                dcl doc2: Doctor := new Doctor("marcelo", 40 ,<CARDIOLOGIA>);

                safetyNet := SafetyNetNetwork`getInstance();

                safetyNet.addHospital(hos1);
                safetyNet.addHospital(hos2);

                safetyNet.addDoctor(doc1);
                safetyNet.addDoctor(doc2);

                safetyNet.associateDoctorToHospital(hos1.getId(), doc1.getId());
                safetyNet.associateDoctorToHospital(hos2.getId(), doc1.getId());
                safetyNet.associateDoctorToHospital(hos2.getId(), doc2.getId());

                assertEqual(hos1.getDoctorsIds(), {doc1.getId()});
                assertEqual(hos2.getDoctorsIds(), {doc1.getId(),doc2.getId()});

                assertEqual( safetyNet.getDoctors(), {doc1.getId() |-> doc1, doc2.getId() |->doc2});

                safetyNet.clearInstance();

);
```

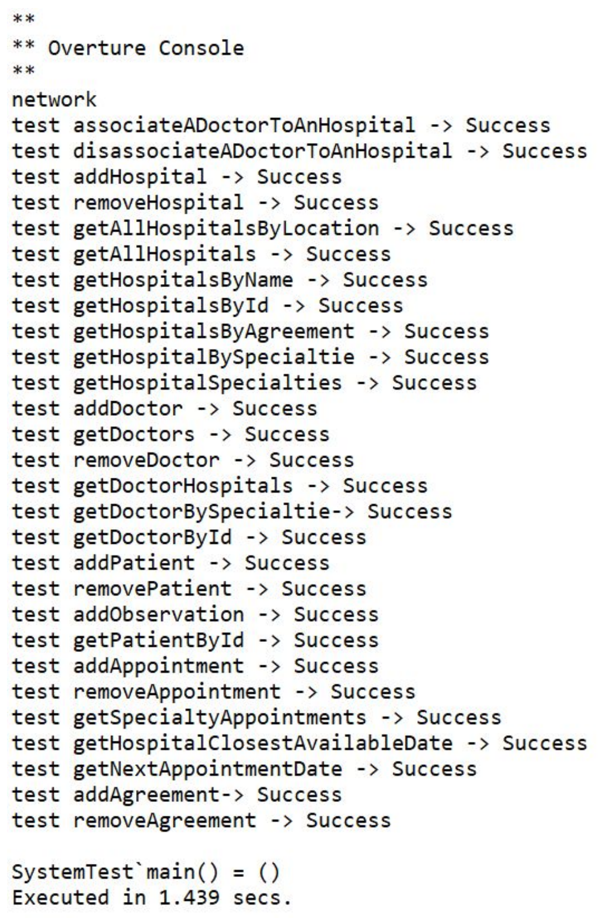
```
-- verifies if the search of a hospital by specialty work as expected
private testGetHospitalBySpecialtie: () ==> ()
        testGetHospitalBySpecialtie () == (

                dcl hos1: Hospital := new Hospital("Hospital Sao Joao",mk_ModelUtils`Location("Porto",
"Alameda Prof. HernÃ¢ni Monteiro","4200-319"), {<ADSE>,<MEDICARE>});
                dcl hos2: Hospital := new Hospital("Hospital da Luz
Lisboa",mk_ModelUtils`Location("Lisboa", "Avenida LusÃada, nÂº 100","4700-959"), {<ADSE>,<MEDIS>,
<MULTICARE>});

                dcl doc1: Doctor := new Doctor("jose",35,<ORTOPEDIA>);
                dcl doc2: Doctor := new Doctor("marcelo", 40 ,<CARDIOLOGIA>);

                dcl aux: set of ModelUtils`Specialty := {<ORTOPEDIA>, <CARDIOLOGIA>, <OFTALMOLOGIA>,
<DERMATOLOGIA>, <GINECOLOGIA>, <NEUROLOGIA>, <PEDIATRIA>, <REUMATOLOGIA>, <UROLOGIA>, <PNEUMOLOGIA>};

                safetyNet := SafetyNetNetwork`getInstance();

                safetyNet.addHospital(hos1);
                safetyNet.addHospital(hos2);

                safetyNet.addDoctor(doc1);
                safetyNet.addDoctor(doc2);

                safetyNet.associateDoctorToHospital(hos1.getId(), doc1.getId());
                safetyNet.associateDoctorToHospital(hos2.getId(), doc1.getId());
                safetyNet.associateDoctorToHospital(hos2.getId(), doc2.getId());

                assertEqual(safetyNet.getHospitalsBySpecialty(<ORTOPEDIA>), {hos1, hos2});
           assertEqual(safetyNet.getHospitalsBySpecialty(<CARDIOLOGIA>), {hos2});

                for all s in set aux do (
                if ((s <> <ORTOPEDIA>) and (s <> <CARDIOLOGIA>)) then
                        assertEqual(safetyNet.getHospitalsBySpecialty(s), {});
                );

                safetyNet.clearInstance();

        );

-- tests if a doctor is correctly disassociated from a hospital
private testDisassociateDoctorToAnHospital: () ==> ()
        testDisassociateDoctorToAnHospital () == (

                dcl hos1: Hospital := new Hospital("Hospital Sao Joao",mk_ModelUtils`Location("Porto",
"Alameda Prof. HernÃ¢ni Monteiro","4200-319"), {<ADSE>,<MEDICARE>});
                dcl hos2: Hospital := new Hospital("Hospital da Luz
Lisboa",mk_ModelUtils`Location("Lisboa", "Avenida LusÃada, nÂº 100","4700-959"), {<ADSE>,<MEDIS>,
<MULTICARE>});

                dcl doc1: Doctor := new Doctor("jose",35,<ORTOPEDIA>);
                dcl doc2: Doctor := new Doctor("marcelo", 40 ,<CARDIOLOGIA>);

                safetyNet := SafetyNetNetwork`getInstance();

                safetyNet.addHospital(hos1);
                safetyNet.addHospital(hos2);

                safetyNet.addDoctor(doc1);
                safetyNet.addDoctor(doc2);
                safetyNet.associateDoctorToHospital(hos1.getId(), doc1.getId());
                safetyNet.associateDoctorToHospital(hos2.getId(), doc1.getId());
                safetyNet.associateDoctorToHospital(hos2.getId(), doc2.getId());


                safetyNet.disassociateDoctorFromHospital(hos2.getId(), doc1.getId());

                assertEqual(hos1.getDoctorsIds(), {doc1.getId()});
                assertEqual(hos2.getDoctorsIds(), {doc2.getId()});

                assertEqual( safetyNet.getDoctors(), {doc1.getId() |-> doc1, doc2.getId() |->doc2});

                safetyNet.clearInstance();

);

-- verifies if the search of a hospital by location work as expected
private testGetHospitalsByLocation: () ==> ()
        testGetHospitalsByLocation () == (

                dcl hos1: Hospital := new Hospital("Hospital Sao Joao",mk_ModelUtils`Location("Porto",
"Alameda Prof. HernÃ¢ni Monteiro","4200-319"), {<ADSE>,<MEDICARE>});
                dcl hos2: Hospital := new Hospital("Hospital da Luz
Lisboa",mk_ModelUtils`Location("Lisboa", "Avenida LusÃada, nÂº 100","4700-959"), {<ADSE>,<MEDIS>,
<MULTICARE>});
                dcl hos3: Hospital := new Hospital("Hospital de Santo
Antonio",mk_ModelUtils`Location("Lisboa", "Avenida de Santo antonio, nï¿½ 300","4750-559"), {<ADSE>});

                safetyNet := SafetyNetNetwork`getInstance();

                safetyNet.addHospital(hos1);
                safetyNet.addHospital(hos2);
                safetyNet.addHospital(hos3);

                assertEqual(safetyNet.getHospitalsByCity("Porto"), {hos1});
                assertEqual(safetyNet.getHospitalsByCity("Lisboa"), {hos2, hos3});

                safetyNet.clearInstance();
);

-- verifies if the search of a hospital by agreement work as expected
private testGetHospitalsByAgreement: () ==> ()
        testGetHospitalsByAgreement () == (

                dcl hos1: Hospital := new Hospital("Hospital Sao Joao",mk_ModelUtils`Location("Porto",
"Alameda Prof. HernÃ¢ni Monteiro","4200-319"), {<ADSE>,<MEDICARE>});
                dcl hos2: Hospital := new Hospital("Hospital da Luz
Lisboa",mk_ModelUtils`Location("Lisboa", "Avenida LusÃada, nÂº 100","4700-959"), {<ADSE>,<MEDIS>,
<MULTICARE>});
                dcl hos3: Hospital := new Hospital("Hospital de Santo
Antonio",mk_ModelUtils`Location("Lisboa", "Avenida de Santo antonio, nï¿½ 300","4750-559"), {<ADSE>});

                safetyNet := SafetyNetNetwork`getInstance();

                safetyNet.addHospital(hos1);
                safetyNet.addHospital(hos2);
                safetyNet.addHospital(hos3);

                assertEqual(safetyNet.getHospitalsByAgreement(<ADSE>), {hos1, hos2, hos3});
                assertEqual(safetyNet.getHospitalsByAgreement(<MEDIS>), {hos2});
                assertEqual(safetyNet.getHospitalsByAgreement(<MULTICARE>), {hos2});
                assertEqual(safetyNet.getHospitalsByAgreement(<MEDICARE>), {hos1});
                safetyNet.clearInstance();
);

-- verifies if the search of a hospital by id work as expected
private testGetHospitalsById: () ==> ()
        testGetHospitalsById () == (

                dcl hos1: Hospital := new Hospital("Hospital Sao Joao",mk_ModelUtils`Location("Porto",
"Alameda Prof. HernÃ¢ni Monteiro","4200-319"), {<ADSE>,<MEDICARE>});
                dcl hos2: Hospital := new Hospital("Hospital da Luz
Lisboa",mk_ModelUtils`Location("Lisboa", "Avenida LusÃada, nÂº 100","4700-959"), {<ADSE>,<MEDIS>,
<MULTICARE>});
                dcl hos3: Hospital := new Hospital("Hospital de Santo
Antonio",mk_ModelUtils`Location("Lisboa", "Avenida de Santo antonio, nï¿½ 300","4750-559"), {<ADSE>});

                safetyNet := SafetyNetNetwork`getInstance();

                safetyNet.addHospital(hos1);
                safetyNet.addHospital(hos2);
                safetyNet.addHospital(hos3);

                assertEqual(safetyNet.getHospitalsById(hos1.getId()), hos1);
                assertEqual(safetyNet.getHospitalsById(hos2.getId()), hos2);
                assertEqual(safetyNet.getHospitalsById(hos3.getId()), hos3);

                safetyNet.clearInstance();
);

-- verifies if the hospitals are obtained correctly through getHospitals
private testGetAllHospitals: () ==> ()
        testGetAllHospitals () == (

                dcl hos1: Hospital := new Hospital("Hospital Sao Joao",mk_ModelUtils`Location("Porto",
"Alameda Prof. HernÃ¢ni Monteiro","4200-319"), {<ADSE>,<MEDICARE>});
                dcl hos2: Hospital := new Hospital("Hospital da Luz
Lisboa",mk_ModelUtils`Location("Lisboa", "Avenida LusÃada, nÂº 100","4700-959"), {<ADSE>,<MEDIS>,
<MULTICARE>});
                dcl hos3: Hospital := new Hospital("Hospital de Santo
Antonio",mk_ModelUtils`Location("Lisboa", "Avenida de Santo antonio, nï¿½ 300","4750-559"), {<ADSE>});

                safetyNet := SafetyNetNetwork`getInstance();

                safetyNet.addHospital(hos1);
                safetyNet.addHospital(hos2);
                safetyNet.addHospital(hos3);

                assertEqual(safetyNet.getHospitals(), {hos1.getId() |-> hos1, hos2.getId() |-> hos2,
hos3.getId() |->hos3});

                safetyNet.clearInstance();
);

-- verifies if the search of a hospital by name work as expected
private testGetHospitalsByName: () ==> ()
        testGetHospitalsByName () == (

                dcl hos1: Hospital := new Hospital("Hospital Sao Joao",mk_ModelUtils`Location("Porto",
"Alameda Prof. HernÃ¢ni Monteiro","4200-319"), {<ADSE>,<MEDICARE>});
                dcl hos2: Hospital := new Hospital("Hospital da Luz
Lisboa",mk_ModelUtils`Location("Lisboa", "Avenida LusÃada, nÂº 100","4700-959"), {<ADSE>,<MEDIS>,
<MULTICARE>});
```

```
                dcl hos3: Hospital := new Hospital("Hospital de Santo
Antonio",mk_ModelUtils`Location("Lisboa", "Avenida de Santo antonio, nï¿½ 300","4750-559"), {<ADSE>});

                safetyNet := SafetyNetNetwork`getInstance();

                safetyNet.addHospital(hos1);
                safetyNet.addHospital(hos2);
                safetyNet.addHospital(hos3);

                assertEqual(safetyNet.getHospitalsByName("Hospital Sao Joao"), {hos1});
                assertEqual(safetyNet.getHospitalsByName("Hospital da Luz Lisboa"), {hos2});
                assertEqual(safetyNet.getHospitalsByName("Hospital de Santo Antonio"), {hos3});

                safetyNet.clearInstance();
);

-- tests if the hospitals where a doctor works are obtained as expected
private testGetDoctorHospitals: () ==> ()
        testGetDoctorHospitals () == (

                dcl hos1: Hospital := new Hospital("Hospital Sao Joao",mk_ModelUtils`Location("Porto",
"Alameda Prof. HernÃ¢ni Monteiro","4200-319"), {<ADSE>,<MEDICARE>});
                dcl hos2: Hospital := new Hospital("Hospital da Luz
Lisboa",mk_ModelUtils`Location("Lisboa", "Avenida LusÃada, nÂº 100","4700-959"), {<ADSE>,<MEDIS>,
<MULTICARE>});

                dcl doc1: Doctor := new Doctor("jose",35,<ORTOPEDIA>);
                dcl doc2: Doctor := new Doctor("marcelo", 40 ,<CARDIOLOGIA>);
                dcl doc3: Doctor := new Doctor("joaquim",50,<CARDIOLOGIA>);

                safetyNet := SafetyNetNetwork`getInstance();

                safetyNet.addHospital(hos1);
                safetyNet.addHospital(hos2);

                safetyNet.addDoctor(doc1);
                safetyNet.addDoctor(doc2);
                safetyNet.addDoctor(doc3);

                safetyNet.associateDoctorToHospital(hos1.getId(), doc1.getId());
                safetyNet.associateDoctorToHospital(hos2.getId(), doc1.getId());
                safetyNet.associateDoctorToHospital(hos2.getId(), doc2.getId());

                assertEqual(safetyNet.getDoctorHospitals(doc1.getId()), {hos1, hos2});
                assertEqual(safetyNet.getDoctorHospitals(doc2.getId()), {hos2});
                assertEqual(safetyNet.getDoctorHospitals(doc3.getId()), {});

                safetyNet.clearInstance();

);


-- verifies if the search of a doctor by specialty work as expected
private testGetDoctorBySpecialtie: () ==> ()
        testGetDoctorBySpecialtie () == (

                dcl doc1: Doctor := new Doctor("jose",35,<ORTOPEDIA>);
                dcl doc2: Doctor := new Doctor("marcelo", 40 ,<CARDIOLOGIA>);
                dcl doc3: Doctor := new Doctor("joaquim",50,<CARDIOLOGIA>);

                safetyNet := SafetyNetNetwork`getInstance();
                safetyNet.addDoctor(doc1);
                safetyNet.addDoctor(doc2);
                safetyNet.addDoctor(doc3);

                assertEqual(safetyNet.getDoctorsBySpecialty(<ORTOPEDIA>), {doc1});
                assertEqual(safetyNet.getDoctorsBySpecialty(<OFTALMOLOGIA>), {});
                assertEqual(safetyNet.getDoctorsBySpecialty(<CARDIOLOGIA>), {doc2, doc3});


                safetyNet.clearInstance();

);

-- verifies if the specialties of a hospital are obtained correctly
private testGetHospitalSpecialties: () ==> ()
        testGetHospitalSpecialties () == (

                dcl hos1: Hospital := new Hospital("Hospital Sao Joao",mk_ModelUtils`Location("Porto",
"Alameda Prof. HernÃ¢ni Monteiro","4200-319"), {<ADSE>,<MEDICARE>});
                dcl hos2: Hospital := new Hospital("Hospital da Luz
Lisboa",mk_ModelUtils`Location("Lisboa", "Avenida LusÃada, nÂº 100","4700-959"), {<ADSE>,<MEDIS>,
<MULTICARE>});

                dcl doc1: Doctor := new Doctor("jose",35,<ORTOPEDIA>);
                dcl doc2: Doctor := new Doctor("marcelo", 40 ,<CARDIOLOGIA>);
                dcl doc3: Doctor := new Doctor("joaquim",50,<CARDIOLOGIA>);

                safetyNet := SafetyNetNetwork`getInstance();

                safetyNet.addHospital(hos1);
                safetyNet.addHospital(hos2);

                safetyNet.addDoctor(doc1);
                safetyNet.addDoctor(doc2);
                safetyNet.addDoctor(doc3);

                safetyNet.associateDoctorToHospital(hos1.getId(), doc1.getId());
                safetyNet.associateDoctorToHospital(hos2.getId(), doc1.getId());
                safetyNet.associateDoctorToHospital(hos2.getId(), doc2.getId());

                assertEqual(safetyNet.getHospitalSpecialties(hos1.getId()), {<ORTOPEDIA>});
                assertEqual(safetyNet.getHospitalSpecialties(hos2.getId()), {<ORTOPEDIA>,
<CARDIOLOGIA>});

                safetyNet.clearInstance();

);

-- verifies if the search of a doctor by id work as expected
private testGetDoctorById: () ==> ()
        testGetDoctorById () == (

                dcl doc1: Doctor := new Doctor("jose",35,<ORTOPEDIA>);
                dcl doc2: Doctor := new Doctor("marcelo", 40 ,<CARDIOLOGIA>);
                dcl doc3: Doctor := new Doctor("joaquim",50,<CARDIOLOGIA>);

                safetyNet := SafetyNetNetwork`getInstance();

                safetyNet.addDoctor(doc1);
                safetyNet.addDoctor(doc2);
                safetyNet.addDoctor(doc3);
                assertEqual(safetyNet.getDoctorById(doc1.getId()), doc1);
                assertEqual(safetyNet.getDoctorById(doc2.getId()), doc2);
                assertEqual(safetyNet.getDoctorById(doc3.getId()), doc3);

                safetyNet.clearInstance();
);

-- verifies if a patient is correctly added to the network
private testAddPatient: () ==> ()
        testAddPatient () == (

                dcl pat1: Patient := new Patient("Susana", 26, "Gripe");
                dcl pat2: Patient := new Patient("Maria", 38, "DoenÃ§a pulmonar");

                safetyNet := SafetyNetNetwork`getInstance();

                safetyNet.addPatient(pat1);
                safetyNet.addPatient(pat2);

                assertEqual(pat1.getName(), "Susana");
                assertEqual(pat2.getName(), "Maria");

                assertEqual(pat1.getAge(), 26);
                assertEqual(pat2.getAge(), 38);

                assertEqual( safetyNet.getPatients(), {pat1.getId() |-> pat1, pat2.getId() |-> pat2});

                safetyNet.clearInstance();

);

-- verifies if a patient is correctly removed from the network
private testRemovePatient: () ==> ()
        testRemovePatient () == (

                dcl pat1: Patient := new Patient("Susana", 26, "Gripe");
                dcl pat2: Patient := new Patient("Maria", 38, "DoenÃ§a pulmonar");

                safetyNet := SafetyNetNetwork`getInstance();

                safetyNet.addPatient(pat1);
                safetyNet.addPatient(pat2);

                safetyNet.removePatient(pat2);

                assertEqual( safetyNet.getPatients(), {pat1.getId() |-> pat1});

                safetyNet.clearInstance();

);

-- verifies if a patient is correctly obtained by his id
private testGetPatientById: () ==> ()
        testGetPatientById () == (

                dcl pat1: Patient := new Patient("Susana", 26, "Gripe");
                dcl pat2: Patient := new Patient("Maria", 38, "DoenÃ§a pulmonar");

                safetyNet := SafetyNetNetwork`getInstance();

                safetyNet.addPatient(pat1);
                safetyNet.addPatient(pat2);
                assertEqual(safetyNet.getPatientById(pat1.getId()), pat1);
                assertEqual(safetyNet.getPatientById(pat2.getId()), pat2);

                safetyNet.clearInstance();
);

-- verifies if an appointment is correctly added, fulfilling all the requirements to create one
private testAddAppointment: () ==> ()
        testAddAppointment () == (

                dcl hos1: Hospital := new Hospital("Hospital Sao Joao",mk_ModelUtils`Location("Porto",
"Alameda Prof. HernÃ¢ni Monteiro","4200-319"), {<ADSE>,<MEDICARE>});
                dcl hos2: Hospital := new Hospital("Hospital da Luz
Lisboa",mk_ModelUtils`Location("Lisboa", "Avenida LusÃada, nÂº 100","4700-959"), {<ADSE>,<MEDIS>,
<MULTICARE>});

                dcl doc1: Doctor := new Doctor("jose",35,<ORTOPEDIA>);
                dcl doc2: Doctor := new Doctor("marcelo", 40 ,<CARDIOLOGIA>);
                dcl doc3: Doctor := new Doctor("joaquim",50,<CARDIOLOGIA>);

                dcl pat1: Patient := new Patient("Susana", 26, "Gripe");
                dcl pat2: Patient := new Patient("Maria", 38, "DoenÃ§a pulmonar");

                dcl dt1: set of ModelUtils`Date := {};
                dcl dt2: set of ModelUtils`Date := {};

                dcl pt1: set of ModelUtils`Date := {};
                dcl pt2: set of ModelUtils`Date := {};

                safetyNet := SafetyNetNetwork`getInstance();

                safetyNet.addHospital(hos1);
                safetyNet.addHospital(hos2);

                safetyNet.addDoctor(doc1);
                safetyNet.addDoctor(doc2);
                safetyNet.addDoctor(doc3);

                safetyNet.addPatient(pat1);
                safetyNet.addPatient(pat2);

                safetyNet.associateDoctorToHospital(hos1.getId(), doc1.getId());
                safetyNet.associateDoctorToHospital(hos2.getId(), doc1.getId());
                safetyNet.associateDoctorToHospital(hos2.getId(), doc2.getId());

                assertEqual(card safetyNet.getDoctorAppointments(doc1.getId()), 0);
                assertEqual(card safetyNet.getDoctorAppointments(doc2.getId()), 0);

                safetyNet.addAppointment(new Appointment(mk_ModelUtils`Date(2018,12,21,8,30),
hos1.getId(), doc1.getId(), pat1.getId()));
                safetyNet.addAppointment(new Appointment(mk_ModelUtils`Date(2018,12,21,8,30),
hos2.getId(), doc2.getId(), pat2.getId()));

                -- doctor appointments
                assertEqual(card safetyNet.getDoctorAppointments(doc1.getId()), 1);
                for all a in set safetyNet.getDoctorAppointments(doc1.getId()) do (
                dt1 := dt1 union {a.getDate()};
                assertEqual(a.getDoctorId(), doc1.getId());
                assertTrue(a.getDate().year > 2017 and a.getDate().month <= 12 and a.getDate().day <
31 and a.getDate().hour < 24 and a.getDate().min < 60);
                );
                assertEqual(card dt1, card safetyNet.getDoctorAppointments(doc1.getId()));

                assertEqual(card safetyNet.getDoctorAppointments(doc2.getId()), 1);
                for all a in set safetyNet.getDoctorAppointments(doc2.getId()) do (
                dt2 := dt2 union {a.getDate()};
                assertEqual(a.getDoctorId(), doc2.getId());
                assertTrue(a.getDate().year > 2017 and a.getDate().month <= 12 and a.getDate().day <
31 and a.getDate().hour < 24 and a.getDate().min < 60);
                );
                assertEqual(card dt2, card safetyNet.getDoctorAppointments(doc2.getId()));


                assertEqual(card safetyNet.getDoctorAppointments(doc3.getId()), 0);


                -- patient appointments
                assertEqual(card safetyNet.getPatientAppointments(pat1.getId()), 1);
                for all a in set safetyNet.getPatientAppointments(pat1.getId()) do (
                pt1 := pt1 union {a.getDate()};
                assertEqual(a.getPatientId(), pat1.getId());
                assertTrue(a.getDate().year > 2017 and a.getDate().month <= 12 and a.getDate().day <
31 and a.getDate().hour < 24 and a.getDate().min < 60);
                );
                assertEqual(card pt1, card safetyNet.getPatientAppointments(pat1.getId()));


                assertEqual(card safetyNet.getPatientAppointments(pat2.getId()), 1);
                for all a in set safetyNet.getPatientAppointments(pat2.getId()) do (
                pt2 := pt2 union {a.getDate()};
                assertEqual(a.getPatientId(), pat2.getId());
                assertTrue(a.getDate().year > 2017 and a.getDate().month <= 12 and a.getDate().day <
31 and a.getDate().hour < 24 and a.getDate().min < 60);
                );
                assertEqual(card pt2, card safetyNet.getPatientAppointments(pat2.getId()));

                -- system
                assertEqual(card safetyNet.getAppointments(), 2);

                -- hospital
            assertEqual(safetyNet.getHospitalNumberOfAppointments(hos1.getId()), 1);
                for all a in set safetyNet.getHospitalAppointments(hos1.getId()) do (
                pt1 := pt1 union {a.getDate()};
                assertEqual(a.getHospitalId(), hos1.getId());
                assertTrue(a.getDoctorId() in set hos1.getDoctorsIds());
                assertTrue(a.getDate().year > 2017 and a.getDate().month <= 12 and a.getDate().day <
31 and a.getDate().hour < 24 and a.getDate().min < 60);
                );

            assertEqual(safetyNet.getHospitalNumberOfAppointments(hos2.getId()), 1);
                for all a in set safetyNet.getHospitalAppointments(hos2.getId()) do (
                pt1 := pt1 union {a.getDate()};
                assertEqual(a.getHospitalId(), hos2.getId());
                assertTrue(a.getDoctorId() in set hos2.getDoctorsIds());
                assertTrue(a.getDate().year > 2017 and a.getDate().month <= 12 and a.getDate().day <
31 and a.getDate().hour < 24 and a.getDate().min < 60);
                );


                safetyNet.clearInstance();

);
```

```
-- verifies if an appointment is correctly removed
private testRemoveAppointment: () ==> ()
        testRemoveAppointment () == (

                dcl hos1: Hospital := new Hospital("Hospital Sao Joao",mk_ModelUtils`Location("Porto",
"Alameda Prof. HernÃ¢ni Monteiro","4200-319"), {<ADSE>,<MEDICARE>});
                dcl hos2: Hospital := new Hospital("Hospital da Luz
Lisboa",mk_ModelUtils`Location("Lisboa", "Avenida LusÃada, nÂº 100","4700-959"), {<ADSE>,<MEDIS>,
<MULTICARE>});

                dcl doc1: Doctor := new Doctor("jose",35,<ORTOPEDIA>);
                dcl doc2: Doctor := new Doctor("marcelo", 40 ,<CARDIOLOGIA>);
                dcl doc3: Doctor := new Doctor("joaquim",50,<CARDIOLOGIA>);

                dcl pat1: Patient := new Patient("Susana", 26, "Gripe");
                dcl pat2: Patient := new Patient("Maria", 38, "DoenÃ§a pulmonar");

                dcl app1: Appointment;
                dcl app2: Appointment;
                dcl app3: Appointment;
                dcl app4: Appointment;

                safetyNet := SafetyNetNetwork`getInstance();

                safetyNet.addHospital(hos1);
                safetyNet.addHospital(hos2);

                safetyNet.addDoctor(doc1);
                safetyNet.addDoctor(doc2);
                safetyNet.addDoctor(doc3);

                safetyNet.addPatient(pat1);
                safetyNet.addPatient(pat2);

                safetyNet.associateDoctorToHospital(hos1.getId(), doc1.getId());
                safetyNet.associateDoctorToHospital(hos2.getId(), doc1.getId());
                safetyNet.associateDoctorToHospital(hos2.getId(), doc2.getId());

                assertEqual(card safetyNet.getDoctorAppointments(doc1.getId()), 0);
                assertEqual(card safetyNet.getDoctorAppointments(doc2.getId()), 0);

                app1 := new Appointment(mk_ModelUtils`Date(2018,12,21,8,30), hos1.getId(),
doc1.getId(), pat1.getId());
                app2 := new Appointment(mk_ModelUtils`Date(2018,12,21,8,30), hos2.getId(),
doc2.getId(), pat2.getId());


                -- add Appointment (app1, app2)
                safetyNet.addAppointment(app1);
                safetyNet.addAppointment(app2);


                -- remove Appointment
                assertTrue(app2 in set safetyNet.getAppointments());

                safetyNet.removeAppointment(app2);

                assertTrue(app2 not in set safetyNet.getAppointments());

                -- verification of data
                -- doctor appointments
                assertEqual(safetyNet.getDoctorAppointments(doc1.getId()), {app1});
                assertEqual(safetyNet.getDoctorAppointments(doc2.getId()), {});
                assertEqual(safetyNet.getDoctorAppointments(doc3.getId()), {});

                -- patient appointments
                assertEqual(safetyNet.getPatientAppointments(pat1.getId()), {app1});
                assertEqual(safetyNet.getPatientAppointments(pat2.getId()), {});

                -- system
                assertEqual(card safetyNet.getAppointments(), 1);
                assertEqual(safetyNet.getAppointments(), {app1});

                -- hospital
         assertEqual(safetyNet.getHospitalNumberOfAppointments(hos1.getId()), 1);
         assertEqual(safetyNet.getHospitalNumberOfAppointments(hos2.getId()), 0);


                -- remove a doctor and all his appointments
                app3 := new Appointment(mk_ModelUtils`Date(2018,01,21,8,30), hos2.getId(),
doc2.getId(), pat1.getId());
                safetyNet.addAppointment(app3);

                safetyNet.removeDoctor(doc2);

                assertEqual(safetyNet.getAppointments(), {app1});


                -- remove a patient and all his appointments

                safetyNet.addDoctor(doc2);

                safetyNet.associateDoctorToHospital(hos2.getId(), doc2.getId());

                app2 := new Appointment(mk_ModelUtils`Date(2018,12,21,8,30), hos2.getId(),
doc2.getId(), pat2.getId());
                app4 := new Appointment(mk_ModelUtils`Date(2018,01,15,8,30), hos2.getId(),
doc1.getId(), pat1.getId());

                safetyNet.addAppointment(app2);
                safetyNet.addAppointment(app4);

                safetyNet.removePatient(pat1);

                assertEqual(safetyNet.getAppointments(), {app2});

                safetyNet.removeAppointment(app2);

                -- remove an hospital and all his appointments

                safetyNet.addPatient(pat1);
                safetyNet.addAppointment(app1);
                safetyNet.addAppointment(app2);
                safetyNet.addAppointment(app3);
                safetyNet.addAppointment(app4);

                assertEqual(safetyNet.getAppointments(), {app1, app2, app3, app4});

                safetyNet.removeHospital(hos2);

                assertEqual(safetyNet.getAppointments(), {app1});
                assertTrue(forall a in set safetyNet.getAppointments() & a.getHospitalId() <>
hos2.getId());

                safetyNet.clearInstance();

);

-- verifies if an agreement is correctly added to a hospital
private testAddAgreement: () ==> ()
        testAddAgreement () == (

                dcl hos1: Hospital := new Hospital("Hospital Sao Joao",mk_ModelUtils`Location("Porto",
"Alameda Prof. HernÃ¢ni Monteiro","4200-319"), {<ADSE>,<MEDICARE>});
                dcl hos2: Hospital := new Hospital("Hospital da Luz
Lisboa",mk_ModelUtils`Location("Lisboa", "Avenida LusÃada, nÂº 100","4700-959"), {<ADSE>,<MEDIS>,
<MULTICARE>});

                safetyNet := SafetyNetNetwork`getInstance();

                safetyNet.addHospital(hos1);
                safetyNet.addHospital(hos2);

                assertEqual(hos1.getAgreements(), {<ADSE>,<MEDICARE>});

                safetyNet.addAgreementToHospital(hos1.getId(),<MULTICARE>);

                assertEqual(hos1.getAgreements(), {<ADSE>,<MEDICARE>, <MULTICARE>});

                safetyNet.clearInstance();

);

-- tests if an agreement is correctly removed from a hospital
private testRemoveAgreement: () ==> ()
        testRemoveAgreement () == (

                dcl hos1: Hospital := new Hospital("Hospital Sao Joao",mk_ModelUtils`Location("Porto",
"Alameda Prof. HernÃ¢ni Monteiro","4200-319"), {<ADSE>,<MEDICARE>});
                dcl hos2: Hospital := new Hospital("Hospital da Luz
Lisboa",mk_ModelUtils`Location("Lisboa", "Avenida LusÃada, nÂº 100","4700-959"), {<ADSE>,<MEDIS>,
<MULTICARE>});

                safetyNet := SafetyNetNetwork`getInstance();

                safetyNet.addHospital(hos1);
                safetyNet.addHospital(hos2);

                safetyNet.removeAgreementFromHospital(hos1.getId(),<ADSE>);

                assertEqual(hos1.getAgreements(), {<MEDICARE>});

                safetyNet.clearInstance();

);

-- checks if an observation is added to the patient observations correctly
private testAddObservation: () ==> ()
        testAddObservation () == (

                dcl pat1: Patient := new Patient("Susana", 26, "Gripe");
                dcl pat2: Patient := new Patient("Maria", 38, "Doenca pulmonar");
                safetyNet := SafetyNetNetwork`getInstance();

                safetyNet.addPatient(pat1);
                safetyNet.addPatient(pat2);

                assertEqual(pat1.getClinicalObservations(), ["Gripe"]);
                assertEqual(pat2.getClinicalObservations(), ["Doenca pulmonar"]);

                safetyNet.addClinicalObservation(pat1.getId(), "Pneumonia");
                safetyNet.addClinicalObservation(pat1.getId(), "Varicela");

                assertEqual(pat1.getClinicalObservations(), ["Gripe", "Pneumonia", "Varicela"]);
                assertEqual(pat2.getClinicalObservations(), ["Doenca pulmonar"]);

                safetyNet.clearInstance();

);

-- verifies if the searchof appointments by specialty, work as expected
private testGetSpecialtyAppointments: () ==> ()
        testGetSpecialtyAppointments () == (

                dcl hos1: Hospital := new Hospital("Hospital Sao Joao",mk_ModelUtils`Location("Porto",
"Alameda Prof. HernÃ¢ni Monteiro","4200-319"), {<ADSE>,<MEDICARE>});
                dcl hos2: Hospital := new Hospital("Hospital da Luz
Lisboa",mk_ModelUtils`Location("Lisboa", "Avenida LusÃada, nÂº 100","4700-959"), {<ADSE>,<MEDIS>,
<MULTICARE>});

                dcl doc1: Doctor := new Doctor("jose",35,<ORTOPEDIA>);
                dcl doc2: Doctor := new Doctor("marcelo", 40 ,<CARDIOLOGIA>);
                dcl doc3: Doctor := new Doctor("joaquim",50,<CARDIOLOGIA>);

                dcl pat1: Patient := new Patient("Susana", 26, "Gripe");
                dcl pat2: Patient := new Patient("Maria", 38, "DoenÃ§a pulmonar");

                dcl app1: Appointment;
                dcl app2: Appointment;
                dcl app3: Appointment;
                dcl app4: Appointment;

                safetyNet := SafetyNetNetwork`getInstance();

                safetyNet.addHospital(hos1);
                safetyNet.addHospital(hos2);

                safetyNet.addDoctor(doc1);
                safetyNet.addDoctor(doc2);
                safetyNet.addDoctor(doc3);

                safetyNet.addPatient(pat1);
                safetyNet.addPatient(pat2);

                safetyNet.associateDoctorToHospital(hos1.getId(), doc1.getId());
                safetyNet.associateDoctorToHospital(hos2.getId(), doc1.getId());
                safetyNet.associateDoctorToHospital(hos2.getId(), doc2.getId());

                app1 := new Appointment(mk_ModelUtils`Date(2018,12,21,8,30), hos1.getId(),
doc1.getId(), pat1.getId());
                app2 := new Appointment(mk_ModelUtils`Date(2018,12,21,8,30), hos2.getId(),
doc2.getId(), pat2.getId());
                app3 := new Appointment(mk_ModelUtils`Date(2018,01,21,8,30), hos2.getId(),
doc2.getId(), pat1.getId());
```

```
                app4 := new Appointment(mk_ModelUtils`Date(2018,01,15,8,30), hos2.getId(),
doc1.getId(), pat1.getId());

                -- add Appointment
                safetyNet.addAppointment(app1);
                safetyNet.addAppointment(app2);
                safetyNet.addAppointment(app3);
                safetyNet.addAppointment(app4);

                assertEqual(safetyNet.getSpecialtyAppointments(<ORTOPEDIA>), {app1, app4});
          assertEqual(safetyNet.getSpecialtyAppointments(<CARDIOLOGIA>), {app2, app3});
          assertEqual(safetyNet.getSpecialtyAppointments(<GINECOLOGIA>), {});


                safetyNet.clearInstance();

);

-- verifies if is obtained the closest date avaivable
private testGetHospitalClosestAvailableDate: () ==> ()
        testGetHospitalClosestAvailableDate () == (

                dcl hos1: Hospital := new Hospital("Hospital Sao Joao",mk_ModelUtils`Location("Porto",
"Alameda Prof. HernÃ¢ni Monteiro","4200-319"), {<ADSE>,<MEDICARE>});
                dcl hos2: Hospital := new Hospital("Hospital da Luz
Lisboa",mk_ModelUtils`Location("Lisboa", "Avenida LusÃada, nÂº 100","4700-959"), {<ADSE>,<MEDIS>,
<MULTICARE>});

                dcl doc1: Doctor := new Doctor("jose",35,<ORTOPEDIA>);
                dcl doc2: Doctor := new Doctor("marcelo", 40 ,<CARDIOLOGIA>);
                dcl doc3: Doctor := new Doctor("joaquim",50,<CARDIOLOGIA>);

                dcl pat1: Patient := new Patient("Susana", 26, "Gripe");
                dcl pat2: Patient := new Patient("Maria", 38, "DoenÃ§a pulmonar");

                dcl app1: Appointment;
                dcl app2: Appointment;
                dcl app3: Appointment;
                dcl app4: Appointment;

                dcl res: ModelUtils`Date_DoctorId;

                safetyNet := SafetyNetNetwork`getInstance();

                safetyNet.addHospital(hos1);
                safetyNet.addHospital(hos2);

                safetyNet.addDoctor(doc1);
                safetyNet.addDoctor(doc2);
                safetyNet.addDoctor(doc3);

                safetyNet.addPatient(pat1);
                safetyNet.addPatient(pat2);

                safetyNet.associateDoctorToHospital(hos1.getId(), doc1.getId());
                safetyNet.associateDoctorToHospital(hos2.getId(), doc1.getId());
                safetyNet.associateDoctorToHospital(hos2.getId(), doc2.getId());
                safetyNet.associateDoctorToHospital(hos1.getId(), doc3.getId());
                safetyNet.associateDoctorToHospital(hos2.getId(), doc3.getId());
                app1 := new Appointment(mk_ModelUtils`Date(2018,01,01,8,30), hos2.getId(),
doc1.getId(), pat1.getId());
                app2 := new Appointment(mk_ModelUtils`Date(2018,01,01,8,30), hos2.getId(),
doc2.getId(), pat2.getId());
                app3 := new Appointment(mk_ModelUtils`Date(2018,01,01,9,00), hos1.getId(),
doc1.getId(), pat1.getId());
                app4 := new Appointment(mk_ModelUtils`Date(2018,01,01,9,30), hos2.getId(),
doc2.getId(), pat2.getId());


                -- add Appointment
                safetyNet.addAppointment(app1);
                safetyNet.addAppointment(app2);
                safetyNet.addAppointment(app3);
                safetyNet.addAppointment(app4);


                res := safetyNet.getClosestAvailableDate({doc1.getId(), doc2.getId()});
                assertEqual(res, mk_ModelUtils`Date_DoctorId(mk_ModelUtils`Date(2018,1,1,9,00),
doc2.getId()));

                res := safetyNet.getClosestAvailableDate({doc1.getId(), doc2.getId(),doc3.getId()});

                assertEqual(res, mk_ModelUtils`Date_DoctorId(mk_ModelUtils`Date(2018,1,1,8,30),
doc3.getId()));


                safetyNet.clearInstance();
);

-- verifies if the next appointment is obtained correctly. It is considered that all appointments have
the duration of 30 min
private testGetNextAppointmentDate: () ==> ()
        testGetNextAppointmentDate () == (

                -- next day
         assertEqual(Appointment`getNextAppointmentDate(mk_ModelUtils`Date(2018,12,12,23,45))
,mk_ModelUtils`Date(2018,12,13,0,15));
                -- next year
         assertEqual(Appointment`getNextAppointmentDate(mk_ModelUtils`Date(2018,12,30,23,45))
,mk_ModelUtils`Date(2019,1,1,0,15));
                -- next month
         assertEqual(Appointment`getNextAppointmentDate(mk_ModelUtils`Date(2018,10,30,23,45))
,mk_ModelUtils`Date(2018,11,1,0,15));
                -- next 30 min
         assertEqual(Appointment`getNextAppointmentDate(mk_ModelUtils`Date(2018,10,15,10,30))
,mk_ModelUtils`Date(2018,10,15,11,0));

);

/*

        Tests containing invalid inputs (should be tested one at a time)
*/

--
public testFailForgotToAddDoctor: () ==> ()
        testFailForgotToAddDoctor () == (

                dcl hos1: Hospital := new Hospital("Hospital Sao Joao",mk_ModelUtils`Location("Porto",
"Alameda Prof. HernÃ¢ni Monteiro","4200-319"), {<ADSE>,<MEDICARE>});
                dcl doc1: Doctor := new Doctor("jose",35,<ORTOPEDIA>);
                safetyNet.clearInstance();

                safetyNet := SafetyNetNetwork`getInstance();

                safetyNet.addHospital(hos1);

                -- you can not associate a doctor to an hospital if that doctor was not added to the
network
                        safetyNet.associateDoctorToHospital(hos1.getId(), doc1.getId());

                safetyNet.clearInstance();
);


 public testFailForgotToAddHospital: () ==> ()
        testFailForgotToAddHospital () == (

                dcl hos1: Hospital := new Hospital("Hospital Sao Joao",mk_ModelUtils`Location("Porto",
"Alameda Prof. HernÃ¢ni Monteiro","4200-319"), {<ADSE>,<MEDICARE>});
                dcl doc1: Doctor := new Doctor("jose",35,<ORTOPEDIA>);

                safetyNet.clearInstance();

                safetyNet := SafetyNetNetwork`getInstance();

                safetyNet.addDoctor(doc1);

                -- you can not associate a doctor to an hospital if that hospital was not added to the
network
                        safetyNet.associateDoctorToHospital(hos1.getId(), doc1.getId());

                safetyNet.clearInstance();

        );


 public testFailCanNotRemoveAnAgreement: () ==> ()
        testFailCanNotRemoveAnAgreement () == (

                dcl hos1: Hospital := new Hospital("Hospital Sao Joao",mk_ModelUtils`Location("Porto",
"Alameda Prof. HernÃ¢ni Monteiro","4200-319"), {<ADSE>,<MEDICARE>});

                safetyNet.clearInstance();

                safetyNet := SafetyNetNetwork`getInstance();

                safetyNet.addHospital(hos1);

                -- it is not possible to remove an agreement from an hospital if that agreement didn't
exist already
                        safetyNet.removeAgreementFromHospital(hos1.getId(),<MEDIS>);

                safetyNet.clearInstance();

        );



        public testFailSearchForADoctor: () ==> ()
                testFailSearchForADoctor () == (
                dcl doc1: Doctor := new Doctor("jose",35,<ORTOPEDIA>);
                dcl doc2: Doctor;

                safetyNet.clearInstance();

                safetyNet := SafetyNetNetwork`getInstance();

                -- you can not search for a doctor with an id that does not exist
                doc2 := safetyNet.getDoctorById(doc1.getId());

                safetyNet.clearInstance();

        );


        public testFailDisassociatingADoctorFromAnHospital: () ==> ()
                testFailDisassociatingADoctorFromAnHospital () == (

                dcl hos1: Hospital := new Hospital("Hospital Sao Joao",mk_ModelUtils`Location("Porto",
"Alameda Prof. HernÃ¢ni Monteiro","4200-319"), {<ADSE>,<MEDICARE>});

                dcl doc2: Doctor := new Doctor("marcelo", 40 ,<CARDIOLOGIA>);

                safetyNet.clearInstance();

                safetyNet := SafetyNetNetwork`getInstance();

                -- you can not remove a doctor from an hospital if previously that doctor was not
associated to that hospital
                        safetyNet.disassociateDoctorFromHospital(hos1.getId(), doc2.getId());

                safetyNet.clearInstance();
                );


 public testFailCreatingAnAppointment: () ==> ()
        testFailCreatingAnAppointment () == (
                dcl hos1: Hospital := new Hospital("Hospital Sao Joao",mk_ModelUtils`Location("Porto",
"Alameda Prof. HernÃ¢ni Monteiro","4200-319"), {<ADSE>,<MEDICARE>});

                dcl doc1: Doctor := new Doctor("jose",35,<ORTOPEDIA>);

                dcl pat1: Patient := new Patient("Susana", 26, "Gripe");
                dcl pat2: Patient := new Patient("Maria", 38, "DoenÃ§a pulmonar");

                dcl app1: Appointment;
                dcl app2: Appointment;

                safetyNet.clearInstance();

                safetyNet := SafetyNetNetwork`getInstance();

                safetyNet.addHospital(hos1);

                safetyNet.addDoctor(doc1);

                safetyNet.addPatient(pat1);
                safetyNet.addPatient(pat2);

                safetyNet.associateDoctorToHospital(hos1.getId(), doc1.getId());
                app1 := new Appointment(mk_ModelUtils`Date(2018,12,21,8,30), hos1.getId(), 
doc1.getId(), pat1.getId());
                app2 := new Appointment(mk_ModelUtils`Date(2018,12,21,8,30), hos1.getId(), 
doc1.getId(), pat2.getId());

                -- a doctor and a patient can not have overlapped appointments
                safetyNet.addAppointment(app1);
                safetyNet.addAppointment(app2);

                safetyNet.clearInstance();

        )

functions

traces

end SystemTest
```

## 4.3. Results



```
**
** Overture Console
**
network
test associateADoctorToAnHospital -> Success
test disassociateADoctorToAnHospital -> Success
test addHospital -> Success
test removeHospital -> Success
test getAllHospitalsByLocation -> Success
test getAllHospitals -> Success
test getHospitalsByName -> Success
test getHospitalsById -> Success
test getHospitalsByAgreement -> Success
test getHospitalBySpecialtie -> Success
test getHospitalSpecialties -> Success
test addDoctor -> Success
test getDoctors -> Success
test removeDoctor -> Success
test getDoctorHospitals -> Success
test getDoctorBySpecialtie-> Success
test getDoctorById -> Success
test addPatient -> Success
test removePatient -> Success
test addObservation -> Success
test getPatientById -> Success
test addAppointment -> Success
test removeAppointment -> Success
test getSpecialtyAppointments -> Success
test getHospitalClosestAvailableDate -> Success
test getNextAppointmentDate -> Success
test addAgreement-> Success
test removeAgreement -> Success

SystemTest`main() = ()
Executed in 1.439 secs.
```

| **Test** | **Description** |
|---|---|
| testAssociateDoctorToAnHospital | Verifies if the section related to associate a doctor to an hospital on R11 is correct. |
| testDisassociateDoctorToAnHospital | Verifies if the section related to disassociate a doctor to an hospital on R11 is correct. |
| testAddHospital | Verifies if the section related to add a hospital on R01 is correct. |
| testRemoveHospital | Verifies if the section related to remove a hospital on R01 is correct. |
| testGetHospitalsByLocation | Verifies if the section related to search a hospital by location on R02 is correct. |
| testGetAllHospitals | Verifies if the section related to return all hospitals on R02 is correct. |

| | |
|---|---|
| testGetHospitalsByName | Verifies if the section related to search a hospital by name on R02 is correct. |
| testGetHospitalsById | Verifies if the section related to search a hospital by id on R02 is correct. |
| testGetHospitalsByAgreement | Verifies if the section related to search a hospital by agreement on R02 is correct. |
| testGetHospitalBySpecialty | Verifies if the section related to search a hospital by specialty on R02 is correct. |
| testGetHospitalSpecialties | Verifies if returns the specialties that the hospital have correctly. It is related with R02. |
| testAddDoctor | Verifies if the section related to add a doctor on R03 is correct. |
| testGetAllDoctors | Verifies if the section related to return all doctors on R04 is correct. |
| testRemoveDoctor | Verifies if the section related to remove a doctor on R03 is correct. |
| testGetDoctorHospitals | Verifies if returns the hospitals were the doctor work correctly. It is related with R04. |
| testGetDoctorBySpecialtie | Verifies if the section related to search a doctor by specialty on R04 is correct. |
| testGetDoctorById | Verifies if the section related to search a doctor by id on R04 is correct. |
| testAddPatient | Verifies if the section related to add a patient on R05 is correct. After the insertion it also verifies R06. |
| testRemovePatient | Verifies if the section related to remove a patient on R05 is correct. After removing the patient it also verifies R06. |
| testAddObservation | Verifies R12. |
| testGetPatientById | Verifies if giving the patientId it returns the correct patient and it is related with R06. |
| testAddAppointment | Verifies R07, after adding the appointment verifies if the search of an appointment by doctor, patient and hospital (R09) is correct. |
| testRemoveAppointment | Verifies R08, after removing the appointment verifies if the search of an appointment by doctor, patient and hospital (R09) is correct. |

| | |
|---|---|
| testGetSpecialtyAppointments | Verifies if the section related to search an appointment by specialty on R09 is correct. |
| testGetHospitalClosestAvailableDate | Verifies R10. |
| testGetNextAppointmentDate | Verifies if giving a date it is returned the correct next date available that is related with R07. |
| testAddAgreement | Verifies if the section related to add an agreement on R13 is correct. |
| testRemoveAgreement | Verifies if the section related to remove an agreement on R13 is correct. |

# 5. Model verification

## 5.1. Example of domain verification

One of the proof obligations generated by Overture is:

| No. | PO Name | Type |
|---|---|---|
| 19 | SafetyNetNetwork`getHospitalsById(nat) | legal map application |

The code under analysis (with the relevant map application underlined) is:

```
     --get hospitals by id
public pure getHospitalsById: nat ==> Hospital
   getHospitalsById(hospitalId) == (
      return hospitals(hospitalId);
   )
pre hospitalId in set dom hospitals
post RESULT.getId() = hospitalId;
```

In this case the proof is trivial because the quantification ***"hospitalId in set dom hospitals"*** ensures that the map reference is only accessed inside its domain.

## 5.2. Example of invariant verification

Another proof obligation generated by Overture is:

| No. | PO Name | Type |
|---|---|---|
| 6 | Hospital`addDoctor(nat) | state invariant holds |

The code under analysis (with the relevant state changes underlined) is:

```
 --add doctor
public addDoctor: nat ==> ()
addDoctor(doctorId) == (
   doctorsIds := doctorsIds union {doctorId}
)
pre doctorId not in set doctorsIds and doctorId in set dom
SafetyNetNetwork`getInstance().getDoctors()
post doctorId in set doctorsIds;
```

The relevant invariant under analysis is:

```
inv forall d in set doctorsIds & d in set dom SafetyNetNetwork`getInstance().getDoctors();
```

The method body has the following line:

```
doctorsIds := doctorsIds union {doctorId}
```

We have to prove that after the execution of this line the invariant holds, this means that the set doctorsIds will contain the id's of doctors that are in the instance SafetyNetNetwork:

```
(forall doctorId:nat & ((doctorId not in set doctorsIds) => ((forall d in set doctorsIds & (d in
set (dom (SafetyNetNetwork`getInstance().getDoctors)()))) => (forall d in set doctorsIds & (d in
set (dom (SafetyNetNetwork`getInstance().getDoctors)())))))
```

In this case the state invariant always holds since the doctorId used in the union operation has already been checked in the pre-condition.

# 6. Code generation

After implementing the VDM++ model, we used the code generation feature from overture, that generated almost instantly all the java code related to the model.
Subsequently to the generation we tested all the code, building a command line interface that allowed to play with the generated code and provide a more user friendly interface to verify all the use cases defined in the section 2.1.
The feedback from the group is positive, everything worked fine, out of the box.
The only complain found by the group with this automated generation feature, was the fact that all the invariants, pre-conditions and post-conditions where not verified in the generated methods, which can cause some model consistency issues in the implemented interface.
All the code generated for the interface is contained in the package 'cli' and to execute the interface the user must place the 'cli' package on the folder 'src' generated by overture in the 'project_path/generated/java/src'.
The code as well as a executable jar file can be found in the 'JavaProject' folder inside the project code folder.

# 7. Conclusions

The model that was developed covers all requirements specified on the first topic of this report. This requirements were created based on the specifications of the theme and taking also in account some discussions with the professor. As future work, it would be useful to go deeper on the management of a hospital, adding features like, financial management, user role separation, a review system for doctors and hospitals as well as some other features that may come on the way.

This project was implemented equally by the group members, dividing tasks and discussing ideas to achieve the best possible result:

José Martins - 50%

Marcelo Ferreira - 50 %

This was a great opportunity to understand the influence of a good modeling strategy and how this "type of development" allowed us to accomplish a robust product, without the need to spend a lot of time coding.

# 8. References

1. VDM-10 Language Manual, Peter Gorm Larsen et al, Overture Technical Report Series No. TR-001, March 2014
2. Overture tool web site, http://overturetool.org
3. Materials provided in the 'Métodos Formais da Engenharia de Software' moodle page