

Safety Net Hospital

Métodos Formais em Engenharia de

Software

- 4MIEIC04 –

Filipe Miguel Leitão Ribeiro – [ei11141@fe.up.pt](mailto:ei11141@fe.up.pt)

Gustavo Fernando Marques Duarte de Faria – [up201304501@fe.up.pt](mailto:up201304501@fe.up.pt)

Luis Pedro Silva Couto – [mieic1204994@fe.up.pt](mailto:mieic1204994@fe.up.pt)

Porto, 2017

Indice

[1. Informal system description and list of requirements 3](#_Toc500724217)

[1.1. Informal system description 3](#_Toc500724218)

[1.2. List of requirements 3](#_Toc500724219)

[2. Visual UML model 4](#_Toc500724220)

[3. Formal VDM++ model 5](#_Toc500724221)

[4. Model validation 6](#_Toc500724222)

[5. Model verification 7](#_Toc500724223)

[6. Code generation 8](#_Toc500724224)

[7. Conclusions 9](#_Toc500724225)

[8. References 10](#_Toc500724226)

# Informal system description and list of requirements

## Informal system description

In this project we had to model the information needed to manage a search system for the user seeking services in a Safety net hospital, giving the possibility to schedule an appointment in the desired specialty.

Safety net hospitals provide care for populations which are of low-income, traditionally exploited, receiving public assistance, and uninsured. Safety net hospitals are not defined by their ownership terms; rather they are more devoted to providing the best possible care for those who are barred from health care due to various circumstances.

## List of requirements

|  |  |  |
| --- | --- | --- |
| **Identifier** | **Priority** | **Description** |
| R1 | Mandatory | It should be possible to search for a hospital in the net |
| R2 | Mandatory | It should be possible to search for a doctor in the net |
| R3 | Mandatory | It should be possible to schedule an appointment |
| R4 | Mandatory | It should be possible to get all the hospitals where a given doctor works |
| R5 | Mandatory | It should be possible to register/login a user |
| R6 | Optional | It should be possible to search for the doctor with the lowest expected time of waiting for a specific specialty |
| R7 | Optional | It should be possible to list the specialties covered by the doctors in the hospitals of the net |

# Visual UML model

## Use case model

Uma imagem com verde, céu

Descrição gerada com confiança muito alta

|  |  |
| --- | --- |
| **Use case** | **Description** |
| Register | Basic registration, user must choose a name, username, password and specialty |
| Login | Basic login, user must provide his/her username and password |
| Change Specialty | A user is able to change the specialty he/she defined when he/she registered |
| Search for Doctor | A user is able to search for doctors in the network |
| Search by Name | A user is able to search for doctors by the doctor’s name |
| Search by Hospital | A user is able to search for the doctors which work in a given hospital |
| Search by Specialty | A user is able to search for the doctors whose specialty is the same as the user’s |
| Choose Hospital | A user is able to choose the hospital where he/she wishes to perform the search |
| Choose Doctor | A user is able to choose the doctor he wishes to schedule an appointment with |
| Choose Type of Appointment | A user is able to choose to either schedule a normal appointment or an emergency appointment. |

## Class model

Uma imagem com texto

Descrição gerada com confiança muito alta

|  |  |
| --- | --- |
| **Class** | **Description** |
| Hospitals Net | Represents a network of hospitals. It has lists of hospitals, doctors and users. |
| Hospital | Represents one hospital. It has a list of doctors and the users, and a list of the specialties it provides to its users. |
| Doctor | Represents one doctor. It has a list of the users who are currently seeing this doctor, and it has a name and a specialty. |
| User | Represents one user. It has a name, a specialty that he/she needs, and the information to log into the “system” (username and password). |

# Formal VDM++ model

## Class Hospitals\_net

**class** Hospitals\_net

**types**

**public** String = StringUtils`String;

**instance variables**

**private** hospitals: **set** **of** Hospital := {};

**private** doctors: **set** **of** Doctor := {};

**private** users: **set** **of** User := {};

--ids should always be unique /\*

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

inv not exists d1,d2 in set doctors & d1 <> d2 and d1.getId() = d2.getId();

inv not exists u1, u2 in set users & u1 <> u2 and u1.getId() = u2.getId(); \*/

**operations**

--Hospitals\_net constructor

**public** Hospitals\_net : () ==> Hospitals\_net

Hospitals\_net() == (**return** **self**)

**post** hospitals={} **and** doctors = {} **and** users = {} **and** hospitals = {};

--Get the net's hospitals

**public** getHospitals: () ==> **set** **of** Hospital

getHospitals() == (**return** hospitals);

--Get doctors in the net

**public** getDoctors: () ==> **set** **of** Doctor

getDoctors() == (**return** doctors);

--Get users in the net

**public** getUsers: () ==> **set** **of** User

getUsers() == (**return** users);

--Add a doctor to net

**public** addDoctor: Doctor ==> ()

addDoctor(d) == (doctors := {d} **union** doctors;)

**pre** **not** d **in set** doctors

**post** doctors = {d} **union** doctors~; --the ~ symbol on the variable indicates the value of the state variable before execution of the operation

--Add a user to net

**public** addUser: User ==> ()

addUser(d) == (users := {d} **union** users;)

**pre** **not** d **in set** users

**post** users = {d} **union** users~;

--Add a hospital to net

**public** addHospital: Hospital ==> ()

addHospital(h) == (hospitals := {h} **union** hospitals;)

**pre** **not** h **in set** hospitals

**post** hospitals = {h} **union** hospitals~;

--Get all doctors from a given specialty

**public** getSpecialtyDoctors: String ==> **set** **of** Doctor

getSpecialtyDoctors(s) == (**dcl** ret: **set** **of** Doctor := {}; --var local

**for all** doc **in set** doctors **do**

**if**(doc.getSpecialty() = s) **then**

ret := ret **union** {doc};

**return** ret;);

--Get all hospitals with a given specialty

**public** getHospitalsSpecialty: String ==> **set** **of** Hospital

getHospitalsSpecialty(s) == (

**dcl** ret: **set** **of** Hospital := {};

**for all** hosp **in set** hospitals **do**

**if**(s **in set** hosp.getSpecialties()) **then**

ret := ret **union** {hosp};

**return** ret;);

--Get all hospitals with a given subsystem

**public** getHospitalsSubsystems: String ==> **set** **of** Hospital

getHospitalsSubsystems(s) == (

**dcl** ret: **set** **of** Hospital := {};

**for all** hosp **in set** hospitals **do**

**if**(s **in set** hosp.getSubsystems()) **then**

ret := ret **union** {hosp};

**return** ret;);

--Get doctor with minimum waiting time for a given specialty

**public** getMinWaitDoc: String ==> Doctor

getMinWaitDoc(s) == (

**dcl** sset: **set** **of** Doctor := getSpecialtyDoctors(s);

**dcl** md: Doctor;

**dcl** min: **nat** := 99;

**for all** doc **in set** sset **do**

**if**(doc.waitingTime() < min) **then**

md := doc;

**return** md;);

--Return true if a doctor is in a given hospital

**public** docInHospital: Doctor \* Hospital ==> **bool**

docInHospital(d, h) == (

**for all** doc **in set** h.getDoctors() **do**

**if**(d = doc) **then**

**return** **true**;

**return** **false**;);

--Get the hospitals where a given doctor works

**public** docHospitals: Doctor ==> **set** **of** Hospital

docHospitals(d) == (

**dcl** ret: **set** **of** Hospital := {};

**for all** hosp **in set** hospitals **do**

**if**(docInHospital(d,hosp)) **then**

ret := ret **union** {hosp};

**return** ret;);

**end** Hospitals\_net

## Class Hospital

**class** Hospital

**types**

**public** String = StringUtils`String;

**instance variables**

**private** id: **nat1**;

**private** doctors: **set** **of** Doctor := {};

**private** specialties: **set** **of** String := {};

**private** subsystems: **set** **of** String := {};

--ids should always be unique/\*

inv not exists d1, d2 in set doctors & d1 <> d2 and d1.getId() = d2.getId();

inv not exists u1, u2 in set users & u1 <> u2 and u1.getId() = u2.getId(); \*/

**operations**

--Hospital constructor

**public** Hospital : **nat1** ==> Hospital

Hospital(i) == (

id := i;

**return** **self**)

**post** doctors = {} **and** id = i;

--Get the hospital id

**public** getId: () ==> **nat1**

getId() == (**return** id);

--Get doctors in hospital

**public** getDoctors: () ==> **set** **of** Doctor

getDoctors() == (**return** doctors);

--Add a doctor to hospital

**public** addDoctor: Doctor ==> ()

addDoctor(d) == (doctors := {d} **union** doctors;)

**pre** **not** d **in set** doctors

**post** doctors = {d} **union** doctors~; --the ~ symbol on the variable indicates the value of the state variable before execution of the operation

--Add specialty

**public** addSpecialty: String ==> ()

addSpecialty(s) == (specialties := {s} **union** specialties;)

**pre** **not** s **in set** specialties

**post** specialties = {s} **union** specialties~;

--Get available specialities

**public** getSpecialties: () ==> **set** **of** String

getSpecialties() == (**return** specialties);

--Add subsystem

**public** addSubsystem: String ==> ()

addSubsystem(s) == (subsystems := {s} **union** subsystems;)

**pre** **not** s **in set** subsystems

**post** subsystems = {s} **union** subsystems~;

--Get available subsystems

**public** getSubsystems: () ==> **set** **of** String

getSubsystems() == (**return** subsystems);

**end** Hospital

## Class Doctor

**class** Doctor

**types**

**public** String = StringUtils`String;

**instance variables**

**private** id: **nat1**;

**private** name: String;

**private** specialty: String;

**private** users: **set** **of** User := {};

**operations**

--Doctor constructor

**public** Doctor: **nat1** ==> Doctor

Doctor(i) == (

id := i;

specialty := "generalist";

**return** **self**)

**post** users = {} **and** id = i;

--Get doctor's name

**public** getName: () ==> String

getName() == (**return** name);

--Get doctor's users

**public** getUsers: () ==> **set** **of** User

getUsers() == (**return** users);

--Assign a user to doctor <=> Schedule a patient

**public** addUser: User ==> ()

addUser(p) == (users := {p} **union** users;)

**pre** **not** p **in set** users

**post** users = {p} **union** users~;

--Set doctor name

**public** setName: String ==> ()

setName(n) == (name := n);

--Get doctor id

**public** getId: () ==> **nat1**

getId() == (**return** id);

--Get specialty

**public** getSpecialty: () ==> String

getSpecialty() == (**return** specialty);

--Set specialty

**public** setSpecialty: String ==> ()

setSpecialty(s) == (specialty := s);

--Get Doctor waiting time

**public** waitingTime: () ==> **nat**

waitingTime() == (

**dcl** count: **nat** := 0;

**for all** us **in set** users **do**

(count := count + 1);

**return** count;);

**end** Doctor

## Class User

**class** User

**types**

**public** String = StringUtils`String;

**instance variables**

**private** id: **nat1**;

**private** name: String;

**private** specialty: String;

**private** username: String;

**private** password: String;

**private** loggedIn: **bool** := **false**;

**operations**

--User constructor

**public** User: **nat1** ==> User

User(i) == (id := i; **return** **self**)

**post** id =i;

--Get user id

**public** getId: () ==> **nat1**

getId() == (**return** id);

--Get user name

**public** getName: () ==> String

getName() == (**return** name);

--Get user required specialty

**public** getSpecialty: () ==> String

getSpecialty() == (**return** specialty);

--Get user password

**public** getPassword: () ==> String

getPassword() == (**return** password);

--Get user username

**public** getUsername: () ==> String

getUsername() == (**return** username);

--Get user log in status

**public** getLoggedIn: () ==> **bool**

getLoggedIn() == (**return** loggedIn);

--Set user required specialty

**public** setSpecialty: String ==> ()

setSpecialty(d) == (specialty := d ;)

**post** specialty = d;

--Set user name

**public** setName: String ==> ()

setName(n) == (name := n;)

**post** name = n;

--Sets user password

**public** setPassword: String ==> ()

setPassword(p) == (password:= p;)

**post** password = p;

--Set user username

**public** setUsername: String ==> ()

setUsername(n) == (username := n;)

**post** username = n;

--Set user log in status

**public** setLoggedIn: **bool** ==> ()

setLoggedIn(n) == (loggedIn := n);

--User login

**public** login: String \* String ==> ()

login(e, p) == (

**if**((e = username) **and** (p = password))

**then** setLoggedIn(**true**);)

**pre** loggedIn = **false**;

**end** User

# Model validation

## Class Test\_Hospitals\_net

**class** Test\_Hospitals\_net

**instance variables**

n: Hospitals\_net := new Hospitals\_net();

u: User := new User(1);

uu: User := new User(7);

h: Hospital := new Hospital(2);

hh: Hospital := new Hospital(5);

hhh: Hospital := new Hospital(6);

d: Doctor := new Doctor(3);

dd: Doctor := new Doctor(4);

**operations**

**private** assertTrue: **bool** ==> ()

assertTrue(cond) == return

**pre** cond;

**private** testHospitals\_net: () ==> ()

testHospitals\_net() ==

(

assertTrue(h not **in set** n.getHospitals());

n.addHospital(h);

assertTrue(h in **set** n.getHospitals());

assertTrue(d not **in set** n.getDoctors());

n.addDoctor(d);

assertTrue(d in **set** n.getDoctors());

assertTrue(n.docInHospital(d,h) = false);

h.addDoctor(d);

assertTrue(n.docInHospital(d,h));

assertTrue(u not **in set** n.getUsers());

n.addUser(u);

assertTrue(u in **set** n.getUsers());

assertTrue(d not **in set** n.getSpecialtyDoctors("spec"));

d.setSpecialty("spec");

assertTrue(d in **set** n.getSpecialtyDoctors("spec"));

assertTrue("spec" not **in set** h.getSpecialties());

h.addSpecialty("spec");

assertTrue("spec" in **set** h.getSpecialties());

assertTrue(h in **set** n.getHospitalsSpecialty("spec"));

assertTrue("subs" not **in set** h.getSubsystems());

h.addSubsystem("subs");

assertTrue("subs" in **set** h.getSubsystems());

assertTrue(h in **set** n.getHospitalsSubsystems("subs"));

n.addUser(uu);

d.addUser(uu);

dd.setSpecialty("spec");

assertTrue(d = n.getMinWaitDoc("spec"));

assertTrue(dd <> n.getMinWaitDoc("spec"));

assertTrue(h in **set** n.docHospitals(d));

assertTrue(h.getId() = 2);

d.setName("doc");

assertTrue(d.getName() = "doc");

assertTrue(u not **in set** d.getUsers());

d.addUser(u);

assertTrue(u in **set** d.getUsers());

assertTrue(d.getId() = 3);

assertTrue(u.getId() = 1);

u.setName("user1");

assertTrue(u.getName() = "user1");

u.setSpecialty("spec");

assertTrue(u.getSpecialty() = "spec");

u.setPassword("pass");

assertTrue(u.getPassword() = "pass");

u.setUsername("user");

assertTrue(u.getUsername() = "user");

--u.setLoggedIn(true);

u.login("user", "pass");

assertTrue(u.getLoggedIn()); );

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

main() ==

( **new** Test\_Hospitals\_net().testHospitals\_net(); );

**end** Test\_Hospitals\_net

# Model verification

## Postcondition verification

Using Overture’s feature “Generate Proof Obligation”, one of the proof obligations we get is the following:

|  |  |  |
| --- | --- | --- |
| **No.** | **PO Name** | **Type** |
| 2 | Doctor’addUser(User) | Operation establishes postcondition |

This PO comes from the code in the class Doctor, in the function addUser(User), which is the following:

--Assign a user to doctor <=> Schedule a patient

**public** addUser: User ==> ()

addUser(p) == (

users := {p} **union** users

;)

**pre** **not** p **in set** users

**post** users = {p} **union** users~;

And the Proof Obligation View is the following:

(forall p:User & ((not (p in set users)) => (({p} union users) = ({p} union users))))

In this case, the postcondition is established because the postcondition is the state of the array users and it is established by this function, whether the User passed as an argument is accepted or not. Also, ({p} union users = {p} union users) is an universal truth.

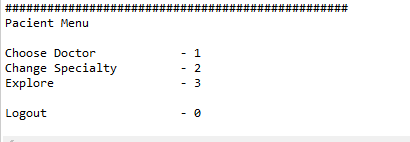
Our model, however, does not provide any Proof Obligations concerning domain verification and invariant verification.

# Code generation

In order to generate the Java code, we used Overture’s feature “Code Generation”. However, this did not generate all the code we needed to run the Java application. The group still had to create the main class, which would allow us to run it.

This class is where we populate the system with some examples of doctors, hospitals, etc., so that the testing of the program can be made without the added hassle of populating the system beforehand.

Then, we developed a console interface for an easier use of the program.



Console Interface

# Conclusions

The results achieved were very positive, as we were able to cover the requirements specified in the guidelines provided and we even achieved to develop some optional requirements suggested by our practical classes’ teacher.

We believe that a way to improve the final product would be to further develop the user interface, whether that be by implementing a graphical interface or by improving the console interface we have at the moment.

# References

* <https://en.wikipedia.org/wiki/Safety_net_hospital>
* <http://overturetool.org>
* <https://moodle.up.pt/pluginfile.php/165033/mod_resource/content/0/VDM%2B%2B.pdf>
* VDM-10 Language Manual, Peter Gorm Larsen et al, Overture Technical Report Series No. TR-001, March 2014