# Report iteration 2 - SWOP

Dieter Geboers, Wouter Schaekers, Stefaan Truijen
-
Professor: Tom Holvoet
Project Advisor: Willem Penninckx

2011-12-31

## 1 Introduction

Let's start off by saying we have the feeling that iteration two went a lot better than iteration one.
After the feedback session of iteration one, Thibault decided to drop out of the course. We were left with only three members. We failed to make the deadline for the report on the testing strategy.
However, we did not let that get to us or let it influence the quality of the report, design and code we made. In fact, we put a maximum amount of effort in doing the assignment as best as we could.

Firstly, we will give you a state of affairs and some numbers about the project. Next, we will discuss the big picture from the current software design we made. We'll specifically elaborate on the Scheduler and everything related to it, seeing as we have put the bigger part of our work in creating a scheduler that is very decent. Then we will list some of the "temporary fixes" we implemented, but that we have planned to be improved due to iteration 3. Finally we'll try to draw a couple of conclusions and give you a brief view on our experience of this iteration.

# Contents

## 2 State of affairs

We have managed to successfully create a design of every aspect of the assignment. The implementation is almost done, but not just yet. Due to a lack of time(which was probably induced by the fact we hadn't brought iteration one to an end), we were forced to implement a couple of things in a "dirty" kind of fashion. We, however, will implement our actual design by the deadline of iteration three for the following things:

1. The way medication items (vitamins, asprin,...) gets handled is currently by using instanceof. We plan to implement a factory pattern here in the future.
2. Machines should also be implemented by using a factory pattern instead of "builders".
3. Treatments aswell, will be implemented with a factory pattern.
4. UnscheduledTaskTest class contains a lot of seemingly random stuff and will be refactored to something cleaner.
5. Documentation may be sloppy, missing or inaccurate for some methods and classes.
6. Not all methods have been implemented defensively. This is a very small minority of methods. Contrary to iteration one, most methods and classes are completely defensive.

Currently the total amount of lines is nearing 20,000 which is enough for a grand total of ... classes! Of those 20,000 lines, Eclemma tells us that ... % is covered. About 3,500 lines of those are in UI. We opted not to test UI with unit tests as the results depend on user input, which is harder than just running them and seeing where it goes wrong, knowing that the underlying system should be monkey proof after extensive unit testing.

We have spent about ... hours averagely per person on this iteration. You can find the total amount of working hours for each person in the table below.

| Dieter | Stefaan | Wouter |
|---|---|---|
| 1337 | 9001 | SO MANY HOURS |

For an even more detailed log of the working hours (containing dates and more information about who was present and what happened at what time), we would like to refer to `http://code.google.com/p/swop-dswx/wiki/WorkingHours` .

## 3 The design

In this chapter we will inform you about the design we've made and will elaborate on some of choices we have made. Firstly, we have modeled our system in 3 layers.

- User interface layer
- Controller layer
- System layer

All communication from the user with the system has to pass through the layer of controllers (same goes the other way around) to assure that our system does what it is supposed to without getting in an inconsistent state.
On the next four pages you will find a class diagram of the system layer of our program. We find this to be the most interesting layer as the controller layer is fairly analogous to what controllers normally look like and as talking about the user interface layer will not inform the reader about what

he/she probably would like to know. In order to read the class diagram as it was originally constructed, we would advise you to put the pages next to eachother from left to right. As big class diagrams tend to not be legible, we'd like to refer to an online version of this diagram just in case: `http://www.student.kuleuven.be/~s0214045/class%20diagram.png` .

## 3.1 Patientfilemanager, patientfile, diagnose, treatment

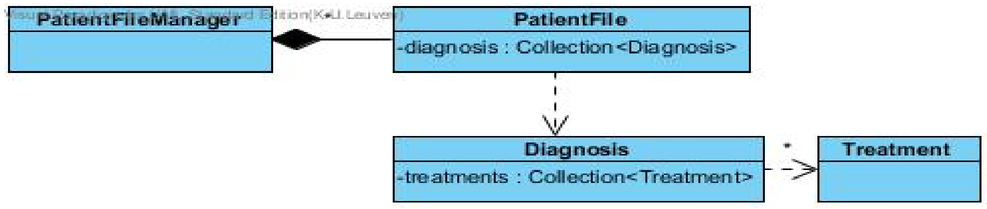
We had a question about where we should store what treament is associated with what diagnosis. In our eyes, there were two possible solutions.

### 3.1.1 Solution

Description: We give PatientFile a field that is a collection of all Diagnosis ever made for this patientfile. Also Diagnosis gets a field that is a collection of all Treatments that have been made for this diagnosis.

Pro: Low coupling, high cohesion and information expert for PatientFile are in good shape. PatientFile only has one dependency and does not need to manage treatments on its own.

Contra: There will be a deep level of function calls if a list of Treatments is needed: from the controller level all the way over to the Treatment level.


PatientFileManager
PatientFile
-diagnosis : Collection<Diagnosis>
Diagnosis
-treatments : Collection<Treatment>
*
Treatment

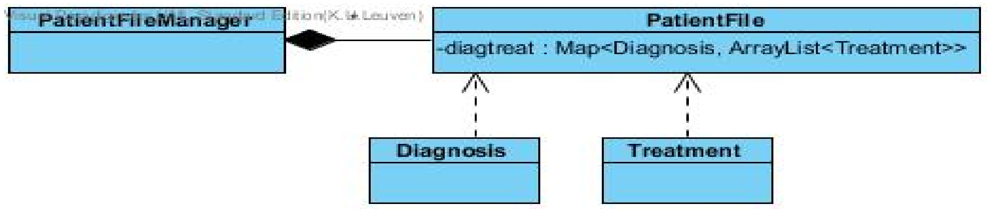
After a meeting with our project advisor, we noticed that in this solution, the GRASP patterns werent violated and the design is arguably elegant. That is why we opted to go with this solution.

### 3.1.2 Alternative solution

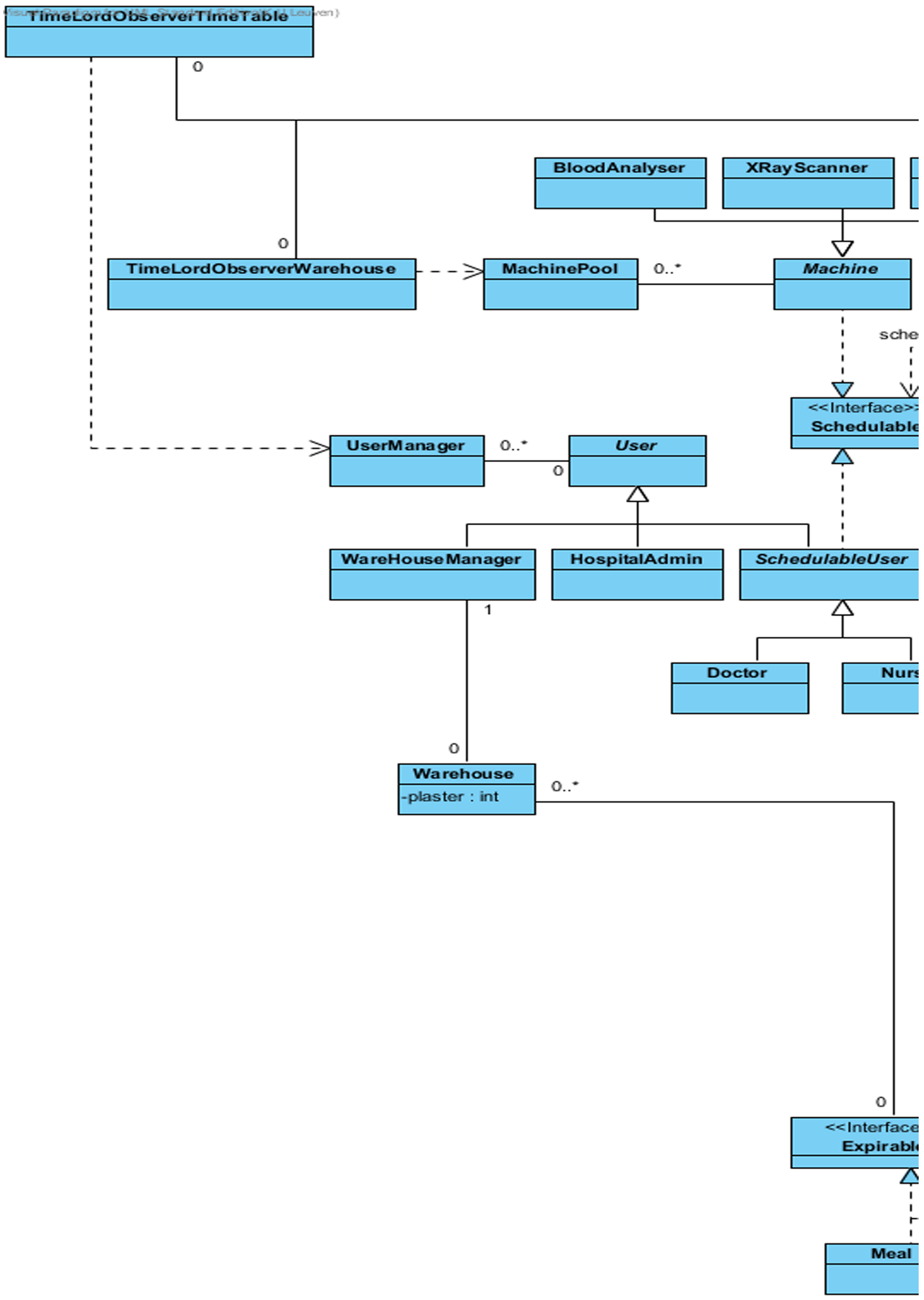
Description: We keep a map from Diagnose to Treatment in PatientFile.

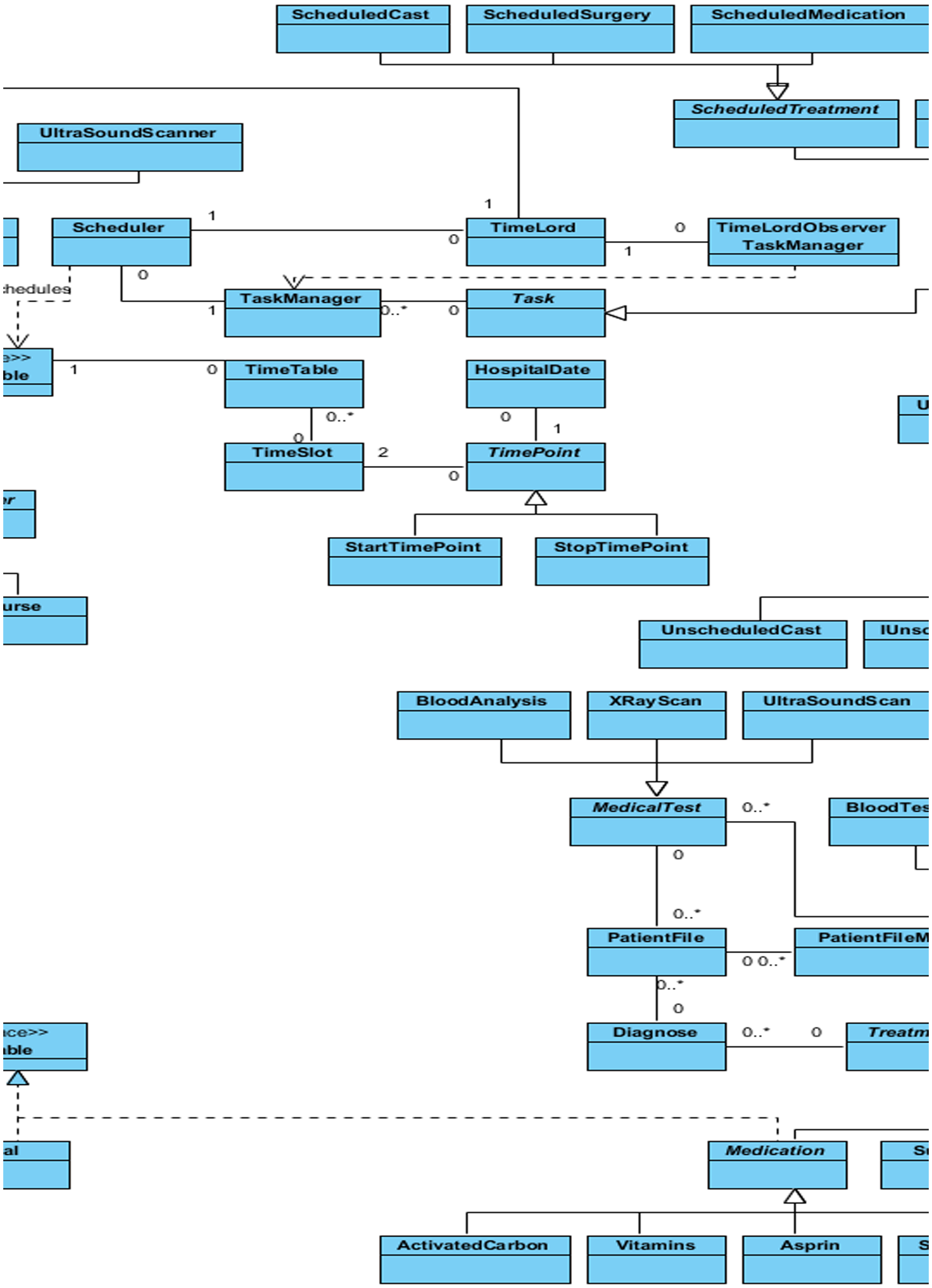
Pro: Easy and efficient access to Treatments.

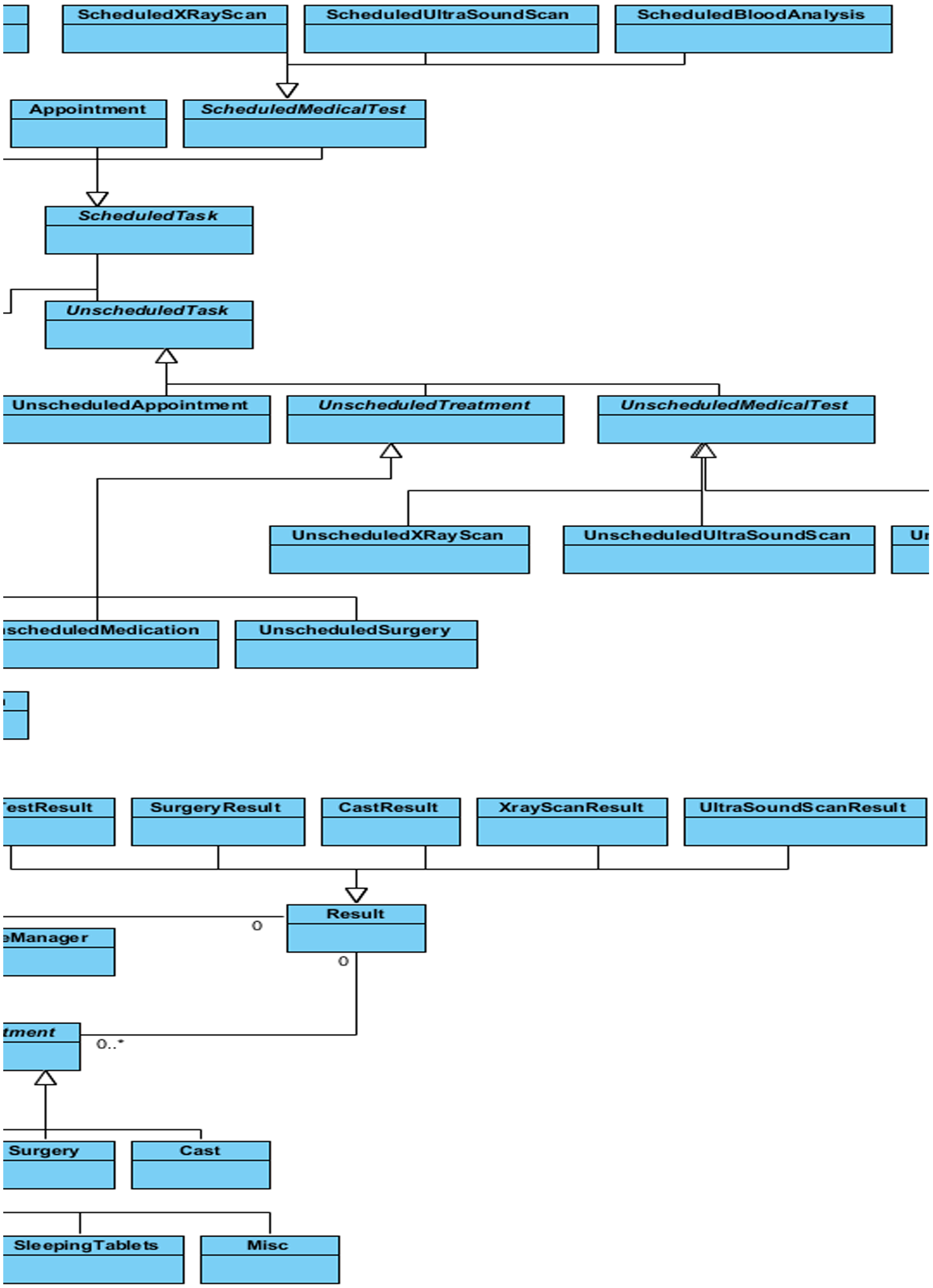
Contra: Gives the patientfile too much responsibility. Also violates the information expert GRASP pattern.

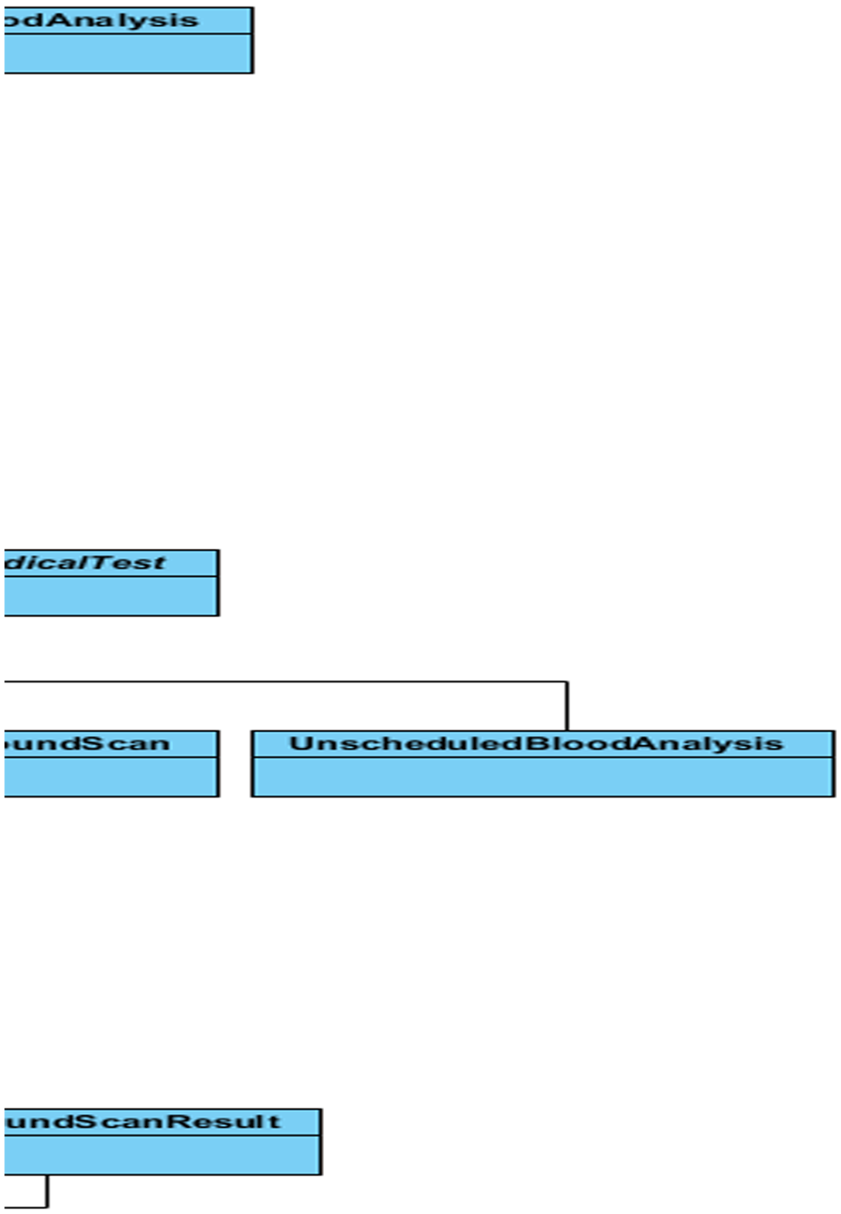


As stated above, we did not chose to implement this solution after noticing that it violated information expert.


TimeLordObserverTimeTable
0
BloodAnalyser
XRayScanner
0
TimeLordObserverWarehouse
MachinePool
0..*
Machine
sche
<<Interface>>
Schedulable
UserManager
0..*
0
User
WareHouseManager
HospitalAdmin
SchedulableUser
1
Doctor
Nurs
0
Warehouse
-plaster : int
0..*
0
<<Interface
Expirabl
Meal


ScheduledCast
ScheduledSurgery
ScheduledMedication
ScheduledTreatment
UltraSoundScanner
Scheduler
TimeLord
TimeLordObserver TaskManager
TaskManager
Task
chedules
ble
TimeTable
HospitalDate
TimeSlot
TimePoint
StartTimePoint
StopTimePoint
urse
UnscheduledCast
IUnsc
BloodAnalysis
XRayScan
UltraSoundScan
MedicalTest
BloodTes
PatientFile
PatientFileM
Diagnose
Treatm
ble
al
Medication
ActivatedCarbon
Vitamins
Asprin
1
0
0..*
2


ScheduledXRayScan
ScheduledUltraSoundScan
ScheduledBloodAnalysis
Appointment
ScheduledMedicalTest
ScheduledTask
UnscheduledTask
UnscheduledAppointment
UnscheduledTreatment
UnscheduledMedicalTest
UnscheduledXRayScan
UnscheduledUltraSoundScan
Ur
scheduledMedication
UnscheduledSurgery
estResult
SurgeryResult
CastResult
XrayScanResult
UltraSoundScanResult
Result
eManager
0
0
tment
0..*
Surgery
Cast
SleepingTablets
Misc


odAnalysis
dicalTest
undScan
UnscheduledBloodAnalysis
undScanResult

## 3.2 MedicalTest, Treatment, Machine and Medication

## 3.3 Current version of Scheduler

### 3.3.1 HospitalDate

### 3.3.2 TimeTable

### 3.3.3 TimePoint

#### StartTimePoint

#### StopTimePoint

### 3.3.4 TimeSlot

### 3.3.5 Schedulable

### 3.3.6 Constraints

### 3.3.7 Task, UnscheduledTask, ScheduledTask, TaskManager

## 3.4 Alternatives and older versions of Scheduler

### 3.4.1 Original version

### 3.4.2 First revision

### 3.4.3 Second revision

# 4 Use cases

# 5 Other relevant changes

## 5.1 Domain protection - not leaking internal data structures

## 5.2 Defensive programming

# 6 Testing and code coverage

# 7 Conclusion and outro