Software Ontwerp
Project Assignment - Iteration III

Contents

1 General Information **3**
1.1 Team Work . . . 3
1.2 The Software . . . 3
1.3 User Interface . . . 3
1.4 JUnit Scenario Test Suite . . . 3
1.5 Domain Model and Refactoring Report . . . 4
1.6 Eclemma Report . . . 4
1.7 The Report . . . 4
1.8 What You Should Hand In . . . 5
1.9 Peer/Self-assessment . . . 6
1.10 Deadlines . . . 6

2 Problem Domain **8**
2.1 Domain Model . . . 8
2.2 Additional Domain Information . . . 8
2.2.1 Priority Cases . . . 8
2.2.2 Hospital Campuses . . . 8
2.2.3 Business Rules . . . 8
2.2.4 Scheduling and Time . . . 9
2.2.5 Stock Management . . . 9
2.2.6 Undoing/Redoing Operations . . . 9

3 Use Cases **11**
3.1 Use Case Diagram . . . 11
3.2 Use Case Descriptions . . . 12
3.2.1 Use Case: Login . . . 12
3.2.2 Use Case: Register Patient . . . 12
3.2.3 Use Case: Consult Patient File . . . 13
3.2.4 Use Case: Close Patient File . . . 13
3.2.5 Use Case: Order Medical Test . . . 13
3.2.6 Use Case: Enter Diagnosis . . . 14
3.2.7 Use Case: Prescribe Treatment . . . 14
3.2.8 Use Case: Approve Diagnosis . . . 15
3.2.9 Use Case: Enter Medical Test Result . . . 16
3.2.10 Use Case: Enter Treatment Result . . . 16
3.2.11 Use Case: Discharge Patient . . . 16
3.2.12 Use Case: Undo Previous Action . . . 17
3.2.13 Use Case: Select Location Preference . . . 17
3.2.14 Use Case: Add Hospital Staff . . . 17
3.2.15 Use Case: Add Hospital Equipment . . . 18
3.2.16 Use Case: Log Out . . . 18

3.2.17 Use Case: Fill Stock in Warehouse 18
3.2.18 Use Case: Advance Time 19
3.2.19 Use Case: List Orders 19

For the course *Software Ontwerp*, you will develop a *Hospital System*, which supports the examination and treatment of patients.

In Section 1, we explain how the project is organized, discuss the quality requirements for the software you will develop and the report you will write. In Section 2, we explain the problem domain of the application, followed by the use cases in Section 3.

For ease of reading, changes in this version of the assignment are marked in red. Be sure to review the whole assignment, in case the meaning of previously defined functionality has changed.

1 General Information

In this section, we explain how the project is organized, what is expected of the software you will develop and the report you will write.

1.1 Team Work

For this project, you will work in groups of four. Each group is assigned an advisor from the educational staff. If you have any questions regarding the project, you can contact your advisor and schedule a meeting. **When you come to the meeting, you are expected to prepare specific questions and have sufficient design documentation available**. It is your own responsibility to organize meetings with your advisor and we advise to do this regularly. Experience from previous years shows that groups that regularly meet with their advisors produce a higher quality design.

If there are problems within the group, you should immediately notify your advisor. Do not wait until right before the deadline or the exam!

1.2 The Software

We expect you to use the development process and the techniques that are taught in this course.

When designing and implementing your system, you should use a defensive programming style. This means that the *user* of the public interface of a class cannot bring the objects of that class, or objects of connected classes, in an inconsistent state.

You are also required to provide extensive class and method documentation, as taught in previous courses.

You are expected to have a public API towards your user interface, which exposes the functionality of the system. Each method exposed in this API must be clearly marked by an annotation. For this API, you should generate a separate documentation page (only including the annotated operations). Naturally, this public API is ideal for creating system-wide test cases. Additionally, you have to validate that your user interface and system-wide test scenarios only use this publicly available API. This validation can be done using a specific tool. The instructions and tools for annotating the API, generating the documentation and checking the use will be made available on Toledo in the course of the third iteration.

You do not have to implement a distributed system. You also do not have to implement networking, use a database, or allow multiple simultaneous logins.

1.3 User Interface

You are strongly recommended to create a text-based user interface, which is considerably simpler to implement than a graphical user interface. The type and looks of the user interface do not determine the grades of your project, only the design and implementation do. Additionally, the amount of time allocated for each iteration does not include the workload for a graphical user interface.

1.4 JUnit Scenario Test Suite

You are supposed to use your public API to create a JUnit scenario test suite, which will be an important part of your demonstration at the examination. These test scenarios approach the system like a user interface (i.e. through system operations) and are only allowed to use functionality exposed by your public API. You are encouraged to create elaborate and interesting scenarios. You will have to hand in a version of this test suite (source code and documentation) together with the domain model and refactoring report (See deadlines).

1.5 Domain Model and Refactoring Report

During this iteration, you are expected to model the extensions in a domain model diagram, as well as provide the accompanying terminology.

Two of the first activities that you should undertake in this third iteration is updating the domain model, and refactoring. You should hand in a report on these activities:

- The report presents an updated domain model and glossary containing a description of any new concepts or old concepts of which the meaning has changed. Unchanged elements do not need to be discussed.
- The report discusses refactorings that were undertaken to change the design and implementation to prepare the design in order to more easily accommodate the new or changed requirements in this iteration. Present this in a structured way, including - per refactoring instance:
  - which objective was considered (code readability, or particular new/changed requirements)
  - which bad smells you detected
  - which refactorings you performed

Also report on your testing approach and how you used your tests during refactoring. This report should not exceed 8 pages (2 pages for domain model/glossary, max. 6 pages on refactoring).

1.6 Eclemma Report

Tests should have good coverage, i.e. a testing strategy that leaves large portions of a software system untested might be of low value. Several tools exist to give a rough estimate of how much code is tested. One of such tools is Eclemma[1]. If this tool reports that only 60% of your code is covered by tests, this might indicate there is a serious problem with (the execution of) your testing strategy. However, be careful when drawing conclusions from both reported high coverage and reported low coverage (understand why you should be careful).

In this iteration, you are expected to use such a tool and briefly report the results in the final report of this iteration.

1.7 The Report

Writing a report is not an easy task. But it is an essential task - it is important that you consider the report to be the first most crucial part of your solution that will be read by your project advisor and teacher. The report should convince the readers of the quality of your design solution as well as of your practical skills (e.g. on UML, testing, etc.). If in doubt, consult your project advisor in an early stage of writing the report.

Remark that the other assets of your solution (code, test suite, extra diagrams) will be studied for further assessment only if the report is good.

1. The baseline objective of the report is to discuss the main design decisions of your design - not to merely provide some diagrams and some accompanying text. Mind that the discussion of your design should be min. 80% of your report!

   Here is one way of reporting a discussion of design decisions [2].

[1] http://www.eclemma.org

[2] Please note that for some crosscutting design issues, you can diverge from this

Discussing design decisions requires describing the context of the decision, by highlighting the use case and the related SSD, and then highlighting the system event that causes system activity that is related to the design decision.

Interaction diagrams then help explain the control flow and responsibilities of classes (make a founded choice between using sequence or collaboration diagrams). In your explanation, do not merely provide a verbatim reproduction of what can be seen in the interaction diagrams, but aim to make a strong case for your design decisions in terms of GRASP and design patterns.

A (part of the) class diagram further helps explain the structure of the solution you are discussing.

Provide a clear motivation for the responsibility assignment that you have used, and support your motivation by discussing alternative solutions.

The UML diagrams that you use should be correct!

Again, only if you make a convincing case for your design in your report, the other assets of your solution will be studied.

2. Next to the discussion of the problem domain and design decisions, the report should include other information (<= 20%), such as

   - a table of contents
   - an introduction that explains the structure of the report
   - a discussion of explicit applications of the GRASP principles, design patterns and refactorings and how they have influenced the design. You can for instance include alternative designs and their trade-offs that have been discussed within the group.
   - a discussion of potential extension scenarios, especially for inheritance hierarchies (e.g. how does the design support an additional subclass?)
   - an explanation of your testing approach, with specific focus for unit tests and system-wide testing using the public API.
   - a critical reflection about the quality of the developed system in its current state. In this section you can highlight the key features of your system, as well as mention points where the system can be improved.
   - information about how you managed the project. Include at least an overview of the tasks performed by each group member, and an estimate of the time invested in those tasks (i.e. time spent per person per task). You should also provide an estimate for the total time that each group member invested in the project.
   - a short test coverage report, based on the output of Eclemma (or another test coverage measuring tool).
   - a conclusion in which you describe the project iteration: problems, interesting experiences,...
   - a full class diagram.

In total, the report is expected to be no more than 25 pages (excl. the full class diagram).

1.8 What You Should Hand In

You hand in 1 physical printout of your report, clearly mentioning the course name, Professor Tom Holvoet's name, the name of your advisor and your group number.

Additionally, you hand in an electronic ZIP-archive via Toledo. **The archive contains the items below and follows the structure defined below**:

- `groupXX` (where XX is your group number (e.g. 01, 12, ...))
  - `design`: a folder containing **all** your design diagrams as image files (PDF, PNG, JPG, ...), including those not used in the report

  - `doc`: a folder containing two versions of your Javadoc documentation
    - `api`: a folder containing the rendered documentation of your public API, which exposes the system's functionality
    - `system`: a folder containing the rendered documentation of your entire system, including internal operations
  - `src`: a folder containing your source code
  - `report.pdf`: a PDF version of your report you handed in
  - `system.jar`: an executable JAR file of your system

When including your source code into the archive, **make sure you use the `svn export` command**, which removes unnecessary repository folders from the source tree.

Make sure you choose relevant file names for your analysis and design diagrams (e.g. `SSDsomeOperation.png`). You do **not** have to include the project file of your UML tool, only the exported diagrams.

We should be able to start your system by executing the JAR file with the following command: `java -jar system.jar`.

1.9 Peer/Self-assessment

In order for you to critically reflect upon the contribution of each team member, you are asked to perform a peer/self-assessment within your team. For each team member (including yourself) and for each of the criteria below, you must give a score on the following scale: *poor/below average/adequate/above average/excellent*. The criteria to be used are:

- Design skills (use of GRASP and DESIGN patterns, ...)
- UML skills (clarity, correctness, ...)
- Coding skills (correctness, defensive programming, documentation,...)
- Testing skills (approach, test suite, coverage, ...)
- Quality of the report (clarity, structure, completeness)
- Collaboration (teamwork, communication, commitment)

In addition to the scores themselves, we expect you to briefly explain for each of the criteria why you have given these particular scores to each of the team members. The total length of your evaluation report must be between 1/2 and 1 full page.

Please be fair and to the point. Your team members will not have access to your evaluation report. If the reports reveal significant problems, the project advisor may discuss these issues with you and/or your team. Please note that your score for this course will be based on the quality of the work that has been delivered, and not on how you are rated by your other team members.

Submit your peer/self-assessment by e-mail to Prof. Tom Holvoet and your project advisor, using the following subject:

[SWOP] peer-/self-assessment of group \$groupnumber\$ by \$firstname\$ \$lastname\$

1.10 Deadlines

- The deadline for handing in the physical report and ZIP-archive on Toledo is **March 30 2012, 16:00**.

  If necessary, a group can apply for an extended deadline (**April 6 2012**), if both the project advisor and Prof. Tom Holvoet are notified by email before **March 22 2012** by the team members.

- The deadline for submitting your peer/self-assessment is **April 6 2012**, by e-mail to both your project advisor and Prof. Tom Holvoet.

- The deadline for sending in your domain model and refactoring report and JUnit scenario test suite (source code and documentation) is **March 5 2012, 18:00**, by e-mail to both your project advisor and Prof. Tom Holvoet.

2 Problem Domain

2.1 Domain Model

It is your task to update the domain model and glossary/terminology section from the previous iteration such that they fit this iteration. For the glossary, it suffices if you include new and changed terminology items. Unchanged items from iteration 2 do not have to be repeated.

2.2 Additional Domain Information

A hospital is run by the *hospital administrator*, who is responsible for managing personnel and equipment. Within the system, the hospital administrator always exists (and is thus able to populate the system with data).

2.2.1 Priority Cases

The situation of some patients may be more urgent than others. Therefore, doctors are able to specify a priority (i.e. normal or urgent) for tests and treatments. The scheduling system will take these priorities into account. If a new test or treatment is scheduled, it is allowed to preempt an already scheduled test or treatment if the new test or treatment has a higher priority. The preempted test or treatment will be rescheduled[3].

2.2.2 Hospital Campuses

In a recent hospital expansion, a second campus has been taken into use. This means that there is a second physical location with its own equipment, warehouse and staff (nurses and warehouse manager). Both campuses are part of the same system, so they share the same data. Patients can (obviously) only be registered at one campus, but can be sent to a different campus for a specific treatment. Doctors work for the hospital, so their appointments can be on both campuses.

Doctors are given the option to select a preference for dealing with the two locations. They can prefer to stay on one campus the first four hours of their shift, and go to the second campus for the next four hours if necessary (they can also stay on one campus for an entire day). The other option is to travel back and forth as needed, but with a maximum of 4 location changes per working day. Unless a doctor specifically chooses his/her own preference (see use case *Select Location Preference*), the *travel back-and-forth* preference is automatically applied.

Travel time between both campuses is 15 minutes, both for patients and for doctors. This time needs to be taken into account when scheduling appointments (e.g. for patient X with appointment A from 08:00 till 08:30 at campus 1, appointment B at campus 2 can only start at 08:45 the earliest).

2.2.3 Business Rules

Within the system, there are several "business rules" that need to be enforced. These rules help to avoid medical mistakes or severe consequences. The rules that need to be enforced are:

- Treatments linked to a diagnosis that requires a second opinion can only be carried out after the diagnosis has received an approving second opinion
- A patient can never be subjected to more than 10 X-ray images per year. When this limit is reached, additional images will have to be scheduled in such a way that this constraint is not violated without waiting longer than necessary.

[3]Note that in a real-life system, starvation would be an issue. For this assignment, you do not have to account for that.

2.2.4 Scheduling and Time

Appointments involving staff members can be scheduled between 9 and 17h, every day of the week.

When a medical test or treatment is scheduled, a nurse needs to be assigned. Within the hospital, a number of nurses are available. The system automatically assigns the new task to the nurse that has the earliest available time span to complete the task.

To simplify the implementation of the system, we assume to be in a time-perfect world, where scheduled tasks take exactly as long as the allocated time span, and never start late. Unless a task can be scheduled back-to-back with another task of the involved staff member (e.g. one task ends at 08:15, the other starts at 08:15), it should start at a full hour (e.g. 08:00 or 09:00).

Contrary to the first iteration, you do have to support advancing time within your system[4]. In order to obtain deterministic behavior, your system has to assume that the **time at startup is November 8th, 2011, 08:00**.

2.2.5 Stock Management

When the stock becomes low, a new order is placed for the stock provider. Since orders take a while to arrive, you should not order supplies twice (e.g. if you have 2 units of plaster today, the system orders 6 more, which arrive the day after tomorrow, but it should not order 6 again tomorrow). Orders are placed automatically and as follows:

1. Normally, stock items are ordered as soon as the amount of stock becomes lower than half of the maximum amount. This happens when the stock becomes low, not if the stock is expected to become low in the future. The amount of stock items ordered equals the maximum amount (capacity of the warehouse) minus the current amount.

2. The hospital always runs out of plaster. To try to solve this, the hospital orders new plaster as soon as there is place in the warehouse. It is possible multiple orders happen in a very short time.

3. For stock items that should be fresh (i.e. food), it is preferred to order regularly instead of waiting a long time and then order a lot. Every day on 23:59 one order is placed. The amount of items ordered equals $15 meals + nbPatients * 3 meals/patient/day * 2 days - nbInStock$, with a minimum of zero, where $nbPatients$ is the number of patients (who are not discharged), $nbInStock$ is the number of items in stock. Note that you should subtract the number of items that will arrive tomorrow, to implement the abovementioned rule that you do not order supplies twice. Also, the system should aim to not order more meals than can be stored on the day of arrival (without taking into account that patients can be discharged in the meantime, which can cause a small storage overload, as mentioned before).

Stock items that expire are removed from the warehouse by the warehouse manager. The warehouse manager does not enter this information into the system since the system knows itself when items expire. For regular stock usage, the system chooses itself which items are taken from the warehouse (using FIFO), this thus does not require user input.

2.2.6 Undoing/Redoing Operations

Doctor's operations can be undone/redone. When an undo occurs, all associated dependencies should be handled appropriately, but the rest of the system should not change. For example, if you undo a *Prescribe Treatment* operation, the system should remove the scheduled treatment from the assigned nurse's schedule, but should not reschedule later appointments (just leave an open timeslot, which can be filled by another appointment). The list of operations that can be undone/redone, and its conditions is as follows:

1. **Order Medical Test / Prescribe Treatment** This operation can only be undone if the test/treatment has not been carried out yet (i.e. the test/treatment has no results).

[4]See use case *Advance Time*

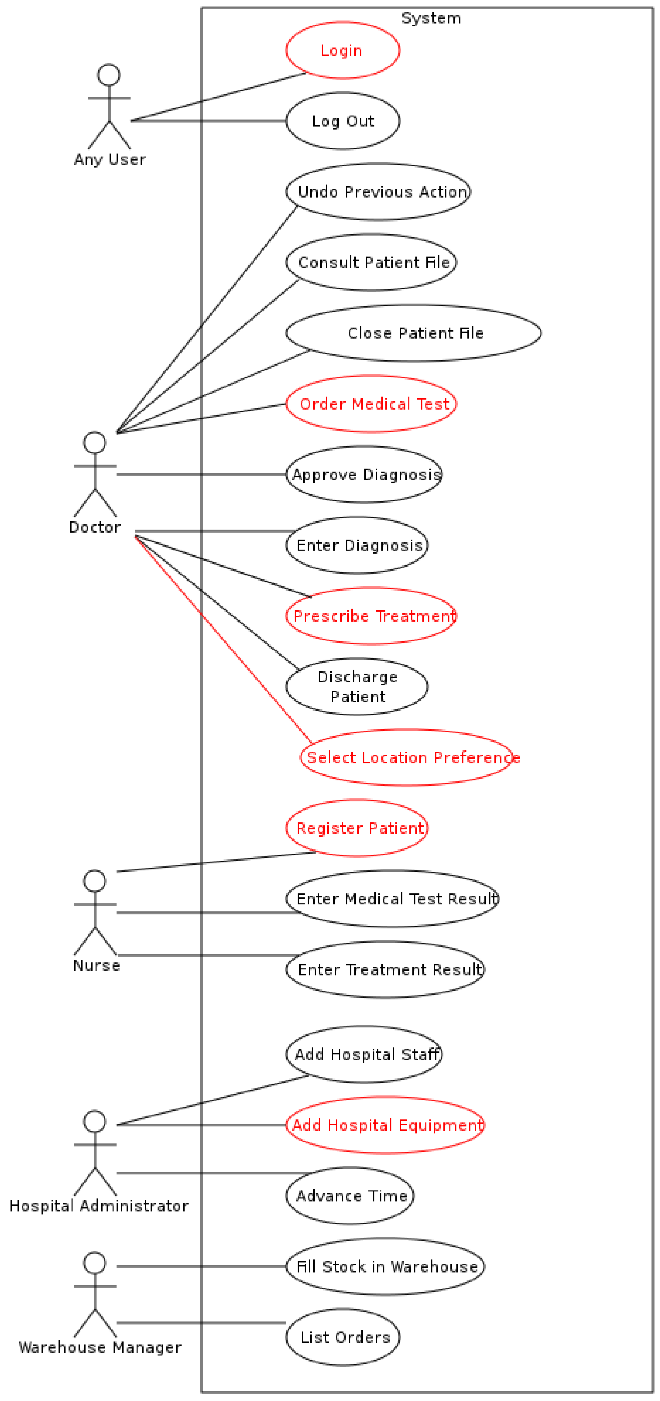
2. **Enter Diagnosis** This operation can only be undone if the diagnosis has not yet been approved or the associated treatment(s) have not been carried out yet

3. **Approve Diagnosis** This operation can only be undone if the associated treatment(s) have not been carried out yet

3 Use Cases

The use cases describe how the actors interact with the system.

3.1 Use Case Diagram



3.2 Use Case Descriptions

3.2.1 Use Case: Login

Primary Actor: Any User

Basic Flow:

1. The user indicates s/he wants to authenticate him/herself to the system
2. The system presents a list of available users and their roles
3. The user selects his/her account from the list
4. The system presents a list of campuses
5. The user selects the campus s/he is currently at
6. The system considers the selected user as the current user at the specified location, until the user logs out

3.2.2 Use Case: Register Patient

Primary Actor: Nurse

Precondition: The use case "Login" has been successfully completed and the user is authenticated as a nurse

Basic Flow:

1. The nurse indicates s/he wants to register a patient who arrives at the check-in desk
2. The system presents a list of previously registered patients
3. The nurse selects the correct patient from the list
4. The system registers the check-in of the patient
5. The nurse selects the doctor that will see the patient
6. The system creates an appointment for the patient with the doctor at the campus where the patient registered, in the next available time slot, at least one hour from the current time[5]
7. The system displays the newly created appointment to the nurse

Alternate Flow:

3. (a) The patient does not occur in the list of previously registered patients (i.e. the patient has not visited the hospital before)
    1. The nurse signals that the patient is new and needs to be created
    2. The system requests the necessary patient details
    3. The nurse enters the requested patient details
    4. The system registers the new patient
    5. *The use case continues with step 4*

Alternate Flow:

2. (a) There is not enough food for the patient for the current day and the next day
    1. The system signals an error
    2. If the other campus would be able to register the patient, the system notifies the nurse, who can redirect the patient
    3. *The use case ends here*

[5]The current time is not hard coded as the startup time, as explained earlier

3.2.3 Use Case: Consult Patient File

Primary Actor: Doctor

Precondition: The use case "Login" has been successfully completed and the user is authenticated as a doctor

Basic Flow:

1. The doctor indicates s/he wants to consult a patient file
2. The system presents the list of patients who are not discharged
3. The doctor selects a patient from the list
4. The system registers the doctor is currently consulting the patient file of the selected patient. If the doctor has already opened another patient file, this previously opened patient file is then closed
5. The system displays a list of all the medical test and treatment results of the selected patient (if any)
6. The doctor can review any of these results in detail

3.2.4 Use Case: Close Patient File

Primary Actor: Doctor

Precondition: The use case "Login" has been successfully completed and the user is authenticated as a doctor

Precondition: The doctor has opened a patient file (see use case "Consult Patient File")

Basic Flow:

1. The doctor indicates s/he wants to close the patient file s/he has opened in the use case "Consult Patient File"
2. The system registers the doctor has closed the patient file

3.2.5 Use Case: Order Medical Test

Primary Actor: Doctor

Precondition: The use case "Login" has been successfully completed and the user is authenticated as a doctor

Precondition: The doctor has opened a patient file (see use case "Consult Patient File")

Precondition: The patient whose file the doctor is working on is not yet discharged

Basic Flow:

1. The doctor indicates s/he wants to order a medical test for the patient whose file s/he is working on
2. The system presents the list of available medical tests
3. The doctor selects the appropriate test from the list
4. The system requests the priority and detailed input for the specific kind of test[6]
5. The doctor enters the requested information

[6] The properties for each test are discussed in the domain model

6. The system stores this information and schedules the requested test. The test must be scheduled in the next time span when all resources for the test are available on either campus, at least one hour from the current time[7]. Preemptive scheduling is possible based on the priority (see above).
7. The system displays the scheduled medical test and its details to the doctor

3.2.6 Use Case: Enter Diagnosis

Primary Actor: Doctor

Precondition: The use case "Login" has been successfully completed and the user is authenticated as a doctor

Precondition: The doctor has opened a patient file (see use case "Consult Patient File")

Precondition: The patient whose file the doctor is working on is not yet discharged

Basic Flow:

1. The doctor indicates s/he wants to enter the diagnosis for the patient whose file s/he is working on
2. The system presents an input form for the diagnosis
3. The doctor enters the diagnosis details
4. The system registers the diagnosis

Alternate Flow:

3. (a) The doctor indicates s/he wants to request a second opinion about the diagnosis
    1. The system presents a list of other doctors
    2. The doctor selects the doctor that needs to give a second opinion
    3. The doctor enters the diagnosis details
    4. The system registers the diagnosis and the request for a second opinion
    5. *The use case ends here*

3.2.7 Use Case: Prescribe Treatment

Primary Actor: Doctor

Precondition: The use case "Login" has been successfully completed and the user is authenticated as a doctor

Precondition: The doctor has opened a patient file (see use case "Consult Patient File")

Precondition: The patient whose file the doctor is working on is not yet discharged

Precondition: A diagnosis has been entered (see use case "Enter Diagnosis") for the patient whose file the doctor is working on

Basic Flow:

1. The doctor indicates s/he wants to prescribe a treatment for the patient whose file s/he is working on
2. The system presents a list of available treatments
3. The doctor selects the appropriate treatment
4. The system requests the priority and detailed input for the specific kind of treatment[8]

[7]The current time is not hard coded as the startup time, as explained earlier

[8]The properties for each treatment are discussed in the domain model

5. The doctor enters the requested information
6. The system stores this information and schedules the requested treatment in the next time span when all resources for the treatment are available on either campus, at least one hour from the current time[9]. Preemptive scheduling is possible based on the priority (see above).
7. The system displays the scheduled treatment and its details to the doctor

Alternate Flow:

6. (a) The treatment belongs to a diagnosis which requires a second opinion, but has not yet been reviewed
    1. The system stores the treatment, but does not yet schedule it
    2. *The use case ends here*

Alternate Flow:

6. (a) There are not enough stock items for this treatment.
    1. The system stores the treatment, but does not yet schedule it. It will be scheduled automatically when there are enough items in stock.
    2. *The use case ends here*

3.2.8 Use Case: Approve Diagnosis

Primary Actor: Doctor

Precondition: The use case "Login" has been successfully completed and the user is authenticated as a doctor

Precondition: The patient whose file the doctor is working on is not yet discharged

Basic Flow:

1. The doctor indicates s/he wants to review a diagnosis
2. The system displays the list of diagnoses which require a second opinion from the current doctor
3. The doctor selects the appropriate diagnosis from the list
4. The system displays the details of the diagnosis and offers the doctor to review the patient's previous test/treatment results
5. The doctor approves the diagnosis that has been determined by his/her colleague
6. The system stores the decision and schedules the treatment associated with the diagnosis
7. The system displays the scheduled treatment and its details to the doctor

Alternate Flow:

5. (a) The doctor does not approve the diagnosis of his/her colleague
    1. The doctor signals his/her disagreement to the system
    2. The system marks the original diagnosis as invalid (the associated treatment is discarded and not scheduled)
    3. The system asks the doctor to enter a new diagnosis: *include use case "Enter Diagnosis"*, but automatically requires the doctor of the original diagnosis to give his/her second opinion about the new diagnosis

[9]The current time is not hard coded as the startup time, as explained earlier

3.2.9 Use Case: Enter Medical Test Result

Primary Actor: Nurse

Precondition: The use case "Login" has been successfully completed and the user is authenticated as a nurse

Basic Flow:

1. A nurse indicates s/he wants to report on a scheduled medical test that has been completed
2. The system displays a list of unfinished[10] medical tests assigned to the current nurse
3. The nurse selects the appropriate medical test
4. The system requests the necessary test information
5. The nurse enters the information
6. The system registers the results

3.2.10 Use Case: Enter Treatment Result

Primary Actor: Nurse

Precondition: The use case "Login" has been successfully completed and the user is authenticated as a nurse

Basic Flow:

1. A nurse indicates s/he wants to report on a scheduled treatment that has been completed
2. The system displays a list of unfinished[11] treatments assigned to the current nurse
3. The nurse selects the appropriate treatment
4. The system requests the necessary test information
5. The nurse enters the information
6. The system registers the results

3.2.11 Use Case: Discharge Patient

Primary Actor: Doctor

Precondition: The use case "Login" has been successfully completed and the user is authenticated as a doctor

Precondition: The doctor has opened a patient file (see use case "Consult Patient File")

Precondition: The patient whose file the doctor is working on is not yet discharged

Basic Flow:

1. A doctor indicates s/he wants to discharge the currently selected patient
2. The system marks the patient as discharged

Alternate Flow:

2. (a) The patient still has unfinished tests or treatments, or an unapproved diagnosis
    1. The system signals an error
    2. *The use case ends here*

[10]Unfinished tests are tests for which no test result has been entered yet
[11]Unfinished treatments are treatments for which no treatment result has been entered yet

3.2.12 Use Case: Undo Previous Action

Primary Actor: Doctor

Precondition: The use case "Login" has been successfully completed and the user is authenticated as a doctor

Basic Flow:

1. A doctor indicates s/he wants to undo one of his/hers previous actions
2. The system shows a list of the last 20 operations that can be undone, as well as a list of the last 5 operations that have been undone
3. The doctor selects an operation to undo, or selects an undone operation to redo (because it should not have been undone)
4. The system executes the requested operation

Alternate Flow:

4. (a) The selected operation cannot be executed successfully
   1. The system signals an error
   2. *The use case ends here*

3.2.13 Use Case: Select Location Preference

Primary Actor: Doctor

Precondition: The use case "Login" has been successfully completed and the user is authenticated as a doctor

Basic Flow:

1. A doctor indicates s/he wants to select a new location preference
2. The system shows the currently selected preference and a list of available preferences
3. The doctor selects the desired preference
4. The system registers the preference and will use it from now when scheduling appointments[12]

3.2.14 Use Case: Add Hospital Staff

Primary Actor: Hospital Administrator

Precondition: The use case "Login" has been successfully completed and the user is authenticated as the hospital administrator

Basic Flow:

1. The hospital administrator indicates s/he wants to add a new staff member
2. The system presents a list of staff member types within the hospital (i.e. nurse, doctor or warehouse manager)
3. The hospital administrator selects the appropriate type of staff member
4. The system requests the necessary staff member details, including the campus if appropriate
5. The hospital administrator provides the requested information
6. The system creates the new staff member

Alternate Flow:

6. (a) The hospital administrator has entered a staff member's name that already exists within the system
   1. The system signals an error

[12]Already scheduled appointments do not change

3.2.15 Use Case: Add Hospital Equipment

Primary Actor: Hospital Administrator

Precondition: The use case “Login” has been successfully completed and the user is authenticated as the hospital administrator

Basic Flow:

1. The hospital administrator indicates s/he wants to add a new machine to the inventory
2. The system presents a list of machine types within the hospital
3. The hospital administrator selects the appropriate type of machine
4. The system requests the necessary machine details, including the campus
5. The hospital administrator provides the requested information
6. The system creates the new machine

Alternate Flow:

6. (a) The hospital administrator has entered a machine identifier that already exists within the system
    1. The system signals an error

3.2.16 Use Case: Log Out

Primary Actor: Any User

Precondition: The use case “Login” has been successfully completed

Basic Flow:

1. The user indicates s/he wants to log out.
2. The system registers the user is logged out.

3.2.17 Use Case: Fill Stock in Warehouse

Primary Actor: Warehouse Manager

Precondition: The use case “Login” has been successfully completed and the user is authenticated as a warehouse manager

Basic Flow:

1. The user indicates new stock items have arrived
2. The system lists the stock categories (e.g. meal, ...)
3. The user selects a category
4. The system lists the orders of the selected category that have not yet been registered as arrived yet
5. The user selects the order that has arrived
6. If the items of the order have an expiration date, the user enters the expiration date[13]
7. The system registers the entered data
8. The user places the stock items in the warehouse.

[13] You can assume all items of one category of one shipping have the same expiration date.

3.2.18 Use Case: Advance Time

Primary Actor: Hospital Administrator

Precondition: The use case "Login" has been successfully completed and the user is authenticated as the hospital administrator

Basic Flow:

1. The user indicates he/she wants to advance the time
2. The user enters a new date and time[14]
3. The system asks the results for every medical test and treatment that is finished before the entered time[15]
4. The user enters the results asked in the previous step
5. The system registers the new date and time

3.2.19 Use Case: List Orders

Primary Actor: Warehouse Manager

Precondition: The use case "Login" has been successfully completed and the user is authenticated as a warehouse manager

Basic Flow:

1. The user indicates he/she wants to see the list of orders
2. The system presents the categories of orders (meals, medication, ...)
3. The user selects one category
4. The system displays the 20 latest placed orders of the selected category that have not arrived already

Good luck!
The SWOP Team members

[14] All automatic actions of the system are performed. This includes ordering meals.
[15] This is highly unrealistic. This use case only serves for demonstration purposes without making the assignment too complex.