Exam Scheduling System

VIA University College

By

Philip Philev

293735

Levente Nagy

293115

Tiberiu Marian

293090

Roberto-Adrian Fat

293099

Supervisors

Mona Wendel Andersen

Steffen Vissing Andresen

Software Technology Engineering

1st Semester

Horsens, Denmark

December 2019

This report is dedicated to *Everyone reading*

Number of Pages: 36

Abstract

An Exam Scheduling System is software application that is designed for a university institution and provides the ability to quickly input desirable data in a coherent manner and generate data files accordingly. In this report an Analysis has been performed based on a set of client Requirement resulting in the construction of a Domain Model and Use Cases. Following is a Design chapter outlying the principles on which the system was build based on the previously set Requirements. An Implementation chapter describes the application of the Design into a software application, consisting of a user interface and dedicated website for displaying the exam schedules. The last chapters test and verify that the Requirements have been met, offer a discussion on the expected results and summarize the content of this report.

Keywords: *Exam, schedule, application, analysis, design, software*

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Scheduling is an important part of every organisation. When the organization in question is a university it becomes much more complex and necessary, as it assures that both students and personnel are assigned to their necessary class, exam, room, etc.

In VIA University College the examination scheduling is done by secretaries, generally using Excel spreadsheets to allocate the students, supervisors and teachers to correct days, timeslots and locations. This task can often prove laborious as the amount of data necessary for the distribution is substantial in addition to possible errors and inaccuracies. As a result, a more efficient method should be designed and implemented in order to handle allocations and reduce the burden of manual input by secretaries. (About VIA, 2019)

This project paper will provide a detailed description on such method by designing and implementing a system capable of taking inputs in the form of students, classes and examiner and assigning them to corresponding exams, rooms and time intervals. Furthermore the system shall generate files that will be displayed in a dedicated website for user convenience.

It is important to state the delimitations set in **Appendix: Project Description**. Firstly, the system developed in this project will not include campus locations other than **Horsens**. Secondly, the input in the system will only include students from the first **4 semesters** of **Software engineering**. Thirdly, the system developed is going to be **Single User System**. Finally, conflict resolution may not be implemented correctly due to some technical issues .

In Chapter 2 the problem domain, domain model and requirements of this project will be stated and discussed.

Chapter 2

Analysis

In order to develop a system which satisfies the stakeholders, the problem needs to be broken down into basic building blocks. As brought up in Background Description found in **Appendix: Project Description** , the current scheduling process is done by using spreadsheets. Observing previously produced examination schedules leads to the identification of the domain objects, determined to be **Students**, **Classes**, **Examiners**, **Rooms** and **Exams** as shown on Figure 2.1.

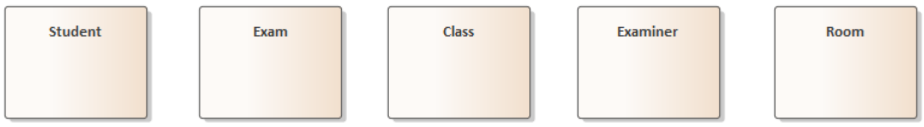


Figure 2.1: Domain Objects

Establishing the basic domain objects leads to the necessity to understand how they cooperate. As it can be inferred from Figure 2.2, **Class** has a **Student** and **Exam** has a participating **Class**, in addition to an **Examiner** and a location in the form of a **Room**.

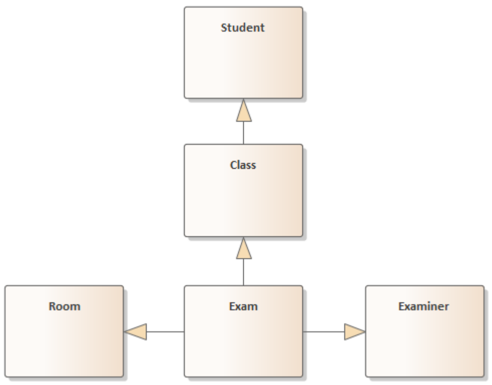


Figure 2.2: Object Cooperation

To further the analysis, it is important to recognize that an **Exam** can be **Written**, **Oral** and **Project** exams. They share similar behavioural patterns, nevertheless a way to differentiate between them needs to be identified. It is logical to treat the **Exam** object as a ground(parent) element defining the similar attributes and operations between the exams, followed by separate objects that inherit the **Exam** properties, but individually handle the specific attributes and operations. A further observation can be made that values for the exam period can be stored in a separate object in contract to each individual object. This results in the formation of the domain model as seen in Figure 2.3: Domain Model.

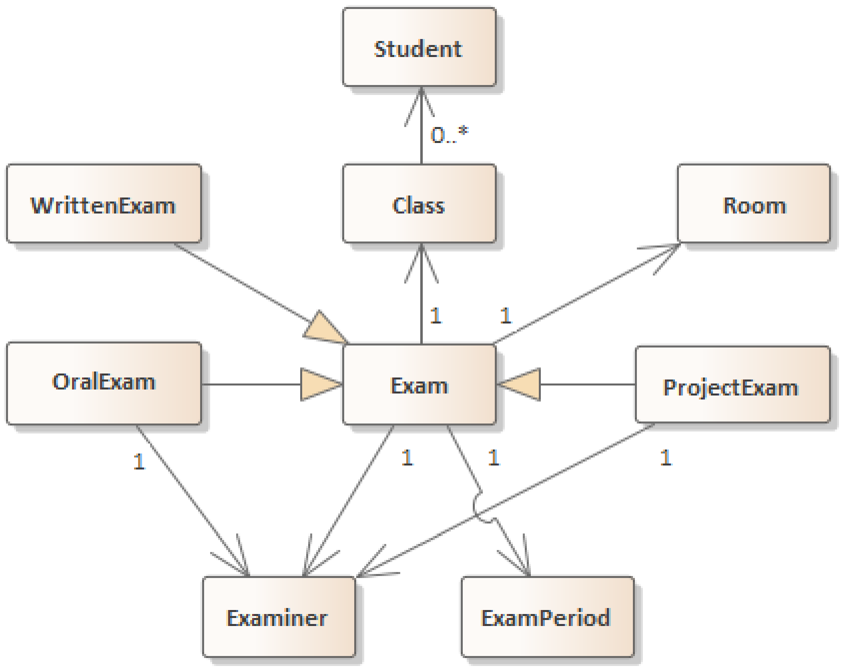


Figure 2.3: Domain Model

The Domain Model will be further expanded after analysis of the requirements and use cases in the following Sections.

2.1 Requirements

In **Appendix: Project description** the background for the problem has been stated. As it currently stands Excel spreadsheets are in use for exams scheduling, which requires a lot of manual processing and data inputs. To develop a system capable of handling the task based on the set delimitations, the users need to be identified and a set of requirements and use cases needs to be specified to achieve a common understanding with the client.

First the actors in the system can be identified as **Secretary** and **Student** or **Examiner**. The **Secretary** is the actor who's role is to input, edit or remove the specified in this chapter domain objects. **Students** and **Examiners** are the actors who have access to the information specified by the **Secretary** and have the role of an end-user.

With this in mind, formulation of the basic requirements from the perspective of the actors with varying priority can be generated as follows:

Critical priority:

1. As a secretary I would like to create an exam schedule.
2. As a secretary I would like to add elements to the exam schedule.
3. As a secretary I would like to save the exam schedule.
4. As a student or examiner, I would like to be able to access the current schedule.

High priority:

5. As a secretary I would like to access previously saved exam schedules.
6. As a secretary I would like to remove elements from the exam schedule.
7. As a secretary I would like to edit elements in the exam schedule.
8. As a secretary I would like to check for conflicts in the exam schedule (where some of the criteria is defined by the institution and some of them is introduced by basic physical restrictions).

Low priority:

9. As a secretary I would like to edit the attributes of the exam period.
10. As a secretary I would like to load previously saved exam schedules.
11. As a student I would like to get a visual representation of the exam schedules.
12. As a student I would like to see my detailed exam schedule for a specific oral or project exam.
13. As a secretary I would like to get an automatically generated schedule for every project and oral exam.

*Where elements are students, classes, examiners, rooms and exams.

2.1.1 Use Case Diagram

Recognizing the actors and formulating the requirements contributes to the construction of a Use Case Diagram shown in Figure 2.4.

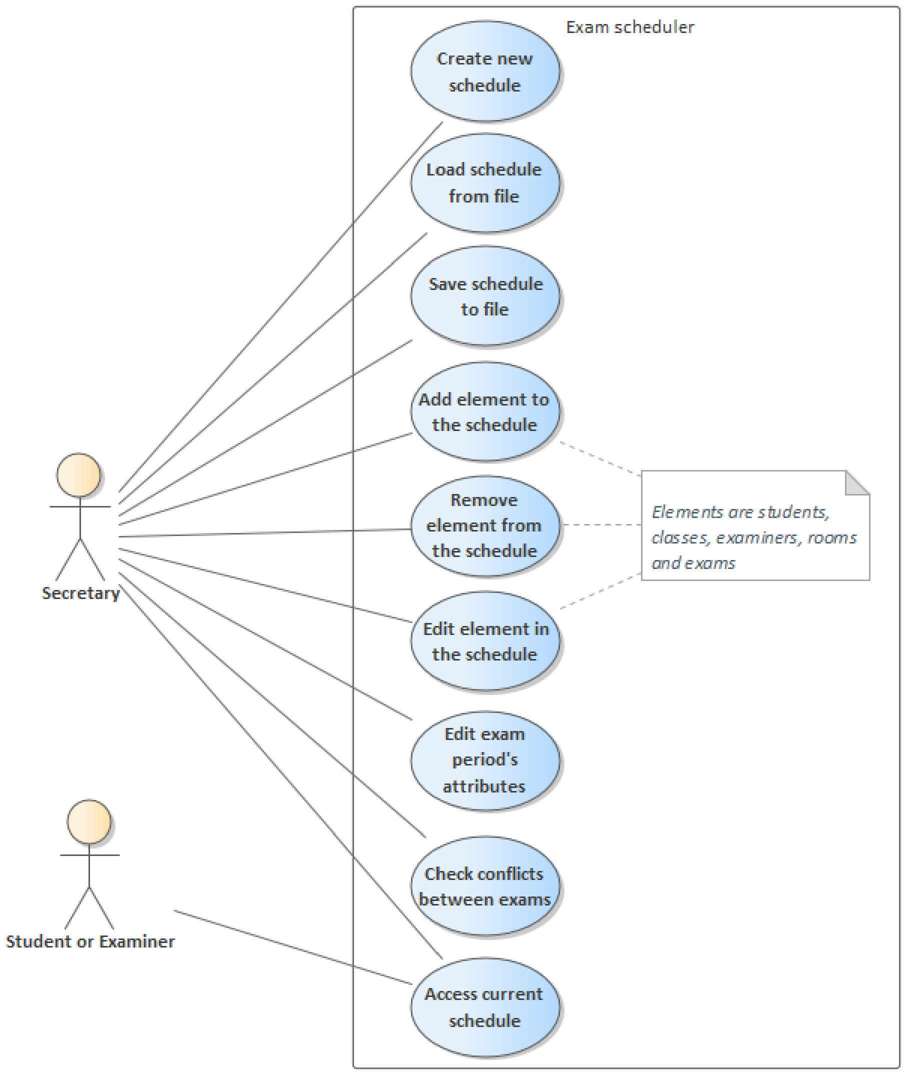
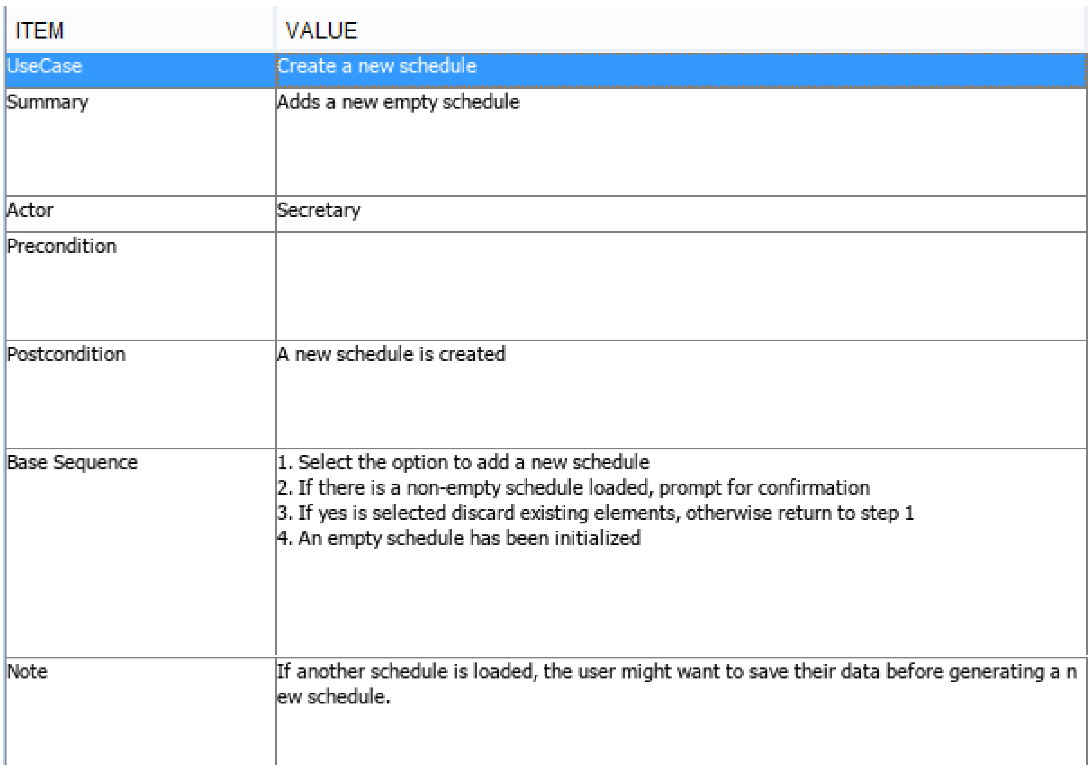

Exam scheduler
Create new schedule
Load schedule from file
Save schedule to file
Add element to the schedule
Remove element from the schedule
Edit element in the schedule
Edit exam period's attributes
Check conflicts between exams
Access current schedule
Elements are students, classes, examiners, rooms and exams
Secretary
Student or Examiner

Figure 2.4: Use Case Diagram

The actors, as shown in the figure, serve their pre-defined roles and interact with specific use cases, which are tailored according to their needs. In the next subsection a more detailed exploration is conducted.

2.1.2 Functional Requirements

As previously stated, actors interact with the use cases. In order to link the functional requirements with their corresponding use case an analysis has been conducted based on the use case descriptions. For the sake of simplicity, only non-trivial and dissimilar use case descriptions have been described.

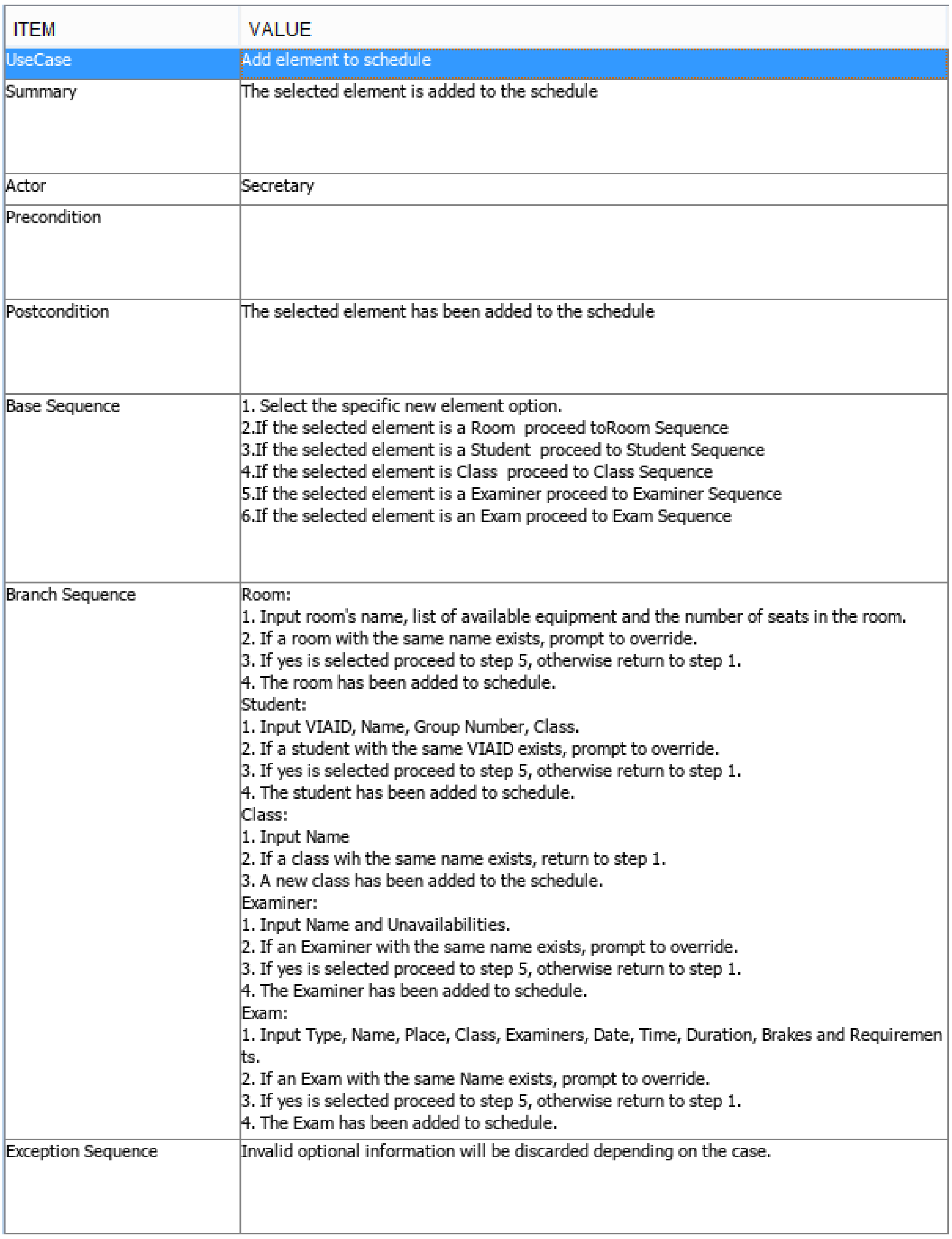


ITEM	VALUE
UseCase	Create a new schedule
Summary	Adds a new empty schedule
Actor	Secretary
Precondition	
Postcondition	A new schedule is created
Base Sequence	1. Select the option to add a new schedule 2. If there is a non-empty schedule loaded, prompt for confirmation 3. If yes is selected discard existing elements, otherwise return to step 1 4. An empty schedule has been initialized
Note	If another schedule is loaded, the user might want to save their data before generating a n ew schedule.

Figure 2.5: "Create new schedule" use case description

Shown in Figure 2.5 is the use case description of "Create new schedule". The actor is clearly the **Secretary** and the following sequence provides the option to add a new schedule with a confirmation check to avoid any mishandling. This use case description covers Requirement 1.

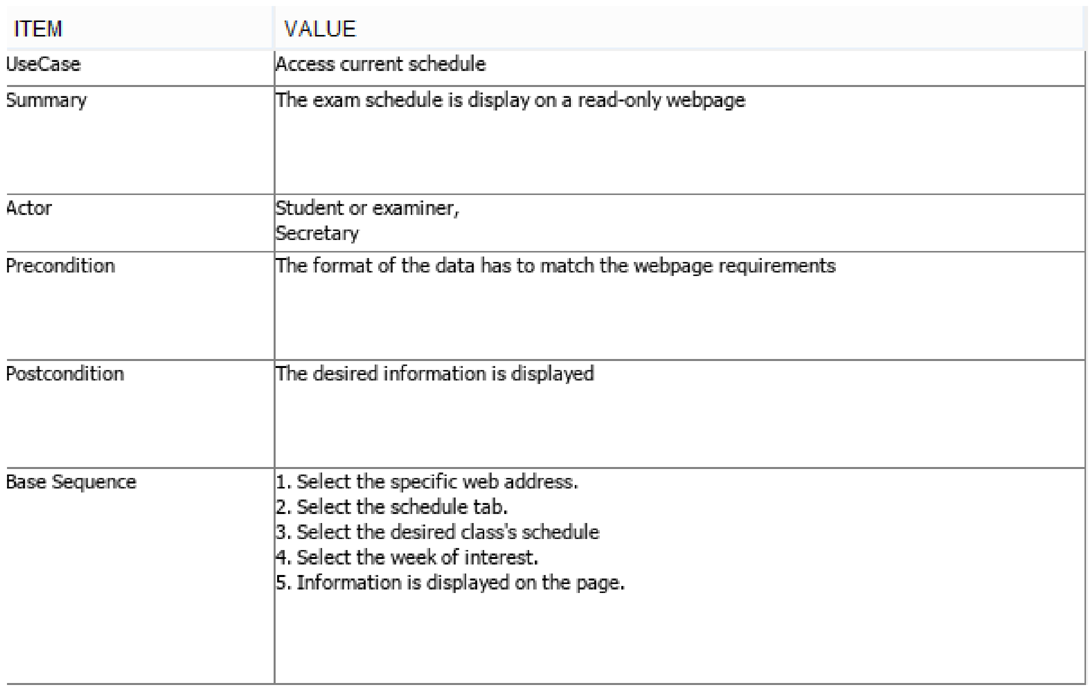
In Figure 2.6 the most detailed use case description is shown, which belongs to use case "Add element to schedule". This use case comprises the elements with the data they handle. These elements are: **Student**, **Class**, **Room**, **Examiner** and **Exam**, with their individual parameters while collectively including Name as an identifier. Every element includes a check for overwriting. This use case description covers Requirement 2.



ITEM	VALUE
UseCase	Add element to schedule
Summary	The selected element is added to the schedule
Actor	Secretary
Precondition	
Postcondition	The selected element has been added to the schedule
Base Sequence	1. Select the specific new element option. 2.If the selected element is a Room proceed toRoom Sequence 3.If the selected element is a Student proceed to Student Sequence 4.If the selected element is Class proceed to Class Sequence 5.If the selected element is a Examiner proceed to Examiner Sequence 6.If the selected element is an Exam proceed to Exam Sequence
Branch Sequence	Room: 1. Input room's name, list of available equipment and the number of seats in the room. 2. If a room with the same name exists, prompt to override. 3. If yes is selected proceed to step 5, otherwise return to step 1. 4. The room has been added to schedule. Student: 1. Input VIAID, Name, Group Number, Class. 2. If a student with the same VIAID exists, prompt to override. 3. If yes is selected proceed to step 5, otherwise return to step 1. 4. The student has been added to schedule. Class: 1. Input Name 2. If a class wih the same name exists, return to step 1. 3. A new class has been added to the schedule. Examiner: 1. Input Name and Unavailabilities. 2. If an Examiner with the same name exists, prompt to override. 3. If yes is selected proceed to step 5, otherwise return to step 1. 4. The Examiner has been added to schedule. Exam: 1. Input Type, Name, Place, Class, Examiners, Date, Time, Duration, Brakes and Requiremen ts. 2. If an Exam with the same Name exists, prompt to override. 3. If yes is selected proceed to step 5, otherwise return to step 1. 4. The Exam has been added to schedule.
Exception Sequence	Invalid optional information will be discarded depending on the case.

Figure 2.6: "Add element to the schedule" use case description

The final use case description is shown on Figure 2.7, describes the use case "Access current schedule and covers Requirements 4,12 and 14. As implied from the use case the Actor, who can either be the Secretary or the Student/Examiner, selects the specific web address and navigates to the Schedule page. After several user specific selections are made the required information is displayed.



ITEM	VALUE
UseCase	Access current schedule
Summary	The exam schedule is display on a read-only webpage
Actor	Student or examiner, Secretary
Precondition	The format of the data has to match the webpage requirements
Postcondition	The desired information is displayed
Base Sequence	1. Select the specific web address. 2. Select the schedule tab. 3. Select the desired class's schedule 4. Select the week of interest. 5. Information is displayed on the page.

Figure 2.7: "Add element to the schedule" use case description

The non-covered uses cases are either trivial, similar or part of the previously covered ones.

2.1.3 Non-Functional Requirements

The following Non-Functional Requirements are of importance when addressing how the system architecture shall be structured. (Eide, 2004)

14. As the customer, I would like to have the surface, where students and examiners can access the current schedule to be a read only webpage without any user handling solutions.
15. As the customer, I would like to have the schedule editing surface to be a privately accessible (only the secretaries can access it) single user system, without any user handling solutions.
16. As the customer, I would like to store the schedule in xml files.

2.1.4 Link between requirements and use cases

In order to fully confirm the relations between the Requirements and the build use cases a link comparison has been shown in Table 2.1.

use case	**covered requirement**
create new schedule	1
load schedule from file	10,5
save schedule to file	3
add element to the schedule	2,13
remove element from the schedule	6
edit element in the schedule	7
edit exam period's attributes	9
check conflicts between exams	8
access current schedule	4,5,11,12

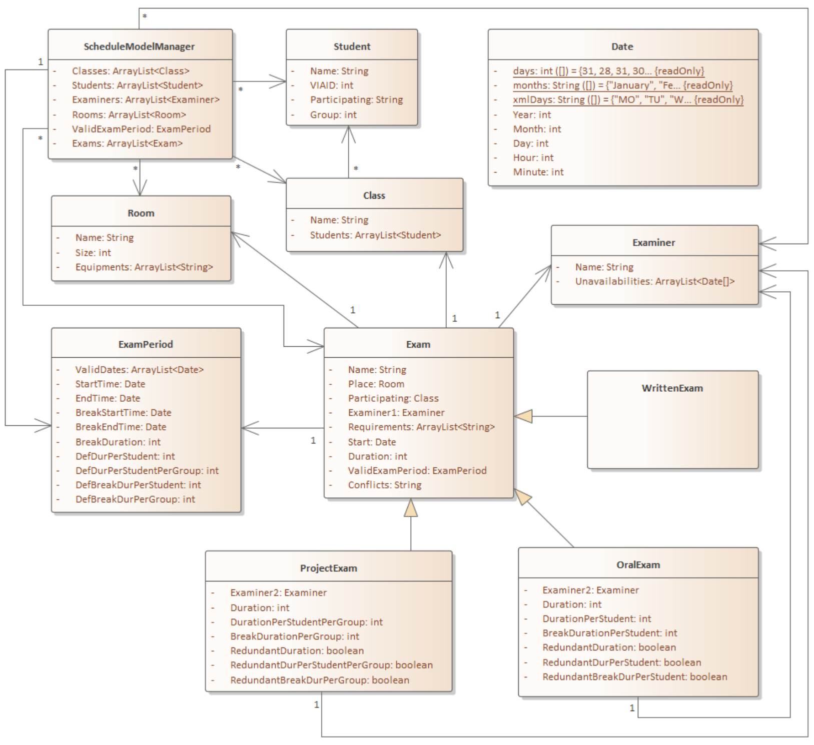
Table 2.1: Use Case - Requirement Table

In the following Chapter: Design, the information provided in the Analysis is extended and used to derive a model for future implementation.

Chapter 3

Design

In this chapter it was natural to extend and use the artefacts of Chapter 2 while modelling the system for simplicity and reduce the margin for error. Thus when the classes' attributes were defined, to achieve the required simplicity, only core information of every class is stored in the attributes, avoiding any redundancy and following the Requirements set in the previous chapter. Furthermore, the classes require a tool that directs data flow between the instances, in the form of a ModelManager, alongside a Date and a Time handling class that can manage precision down to a minute as required by the customer. Considering all of the statements above, the result is seen in Figure 3.1, the Scheduler package's design for the classes .

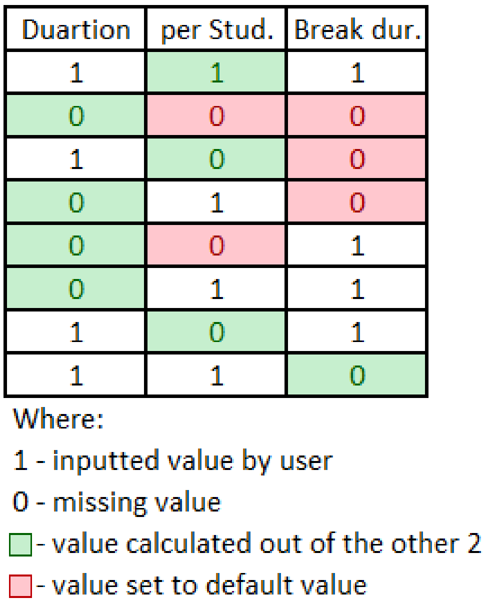


ScheduleModelManager
- Classes: ArrayList<Class>
- Students: ArrayList<Student>
- Examiners: ArrayList<Examiner>
- Rooms: ArrayList<Room>
- ValidExamPeriod: ExamPeriod
- Exams: ArrayList<Exam>
Student
- Name: String
- VIAID: int
- Participating: String
- Group: int
Date
- days: int ([]) = {31, 28, 31, 30... {readOnly}
- months: String ([]) = {"January", "Fe... {readOnly}
- xmlDays: String ([]) = {"MO", "TU", "W... {readOnly}
- Year: int
- Month: int
- Day: int
- Hour: int
- Minute: int
Room
- Name: String
- Size: int
- Equipments: ArrayList<String>
Class
- Name: String
- Students: ArrayList<Student>
Examiner
- Name: String
- Unavailabilities: ArrayList<Date[]>
ExamPeriod
- ValidDates: ArrayList<Date>
- StartTime: Date
- EndTime: Date
- BreakStartTime: Date
- BreakEndTime: Date
- BreakDuration: int
- DefDurPerStudent: int
- DefDurPerStudentPerGroup: int
- DefBreakDurPerStudent: int
- DefBreakDurPerGroup: int
Exam
- Name: String
- Place: Room
- Participating: Class
- Examiner1: Examiner
- Requirements: ArrayList<String>
- Start: Date
- Duration: int
- ValidExamPeriod: ExamPeriod
- Conflicts: String
WrittenExam
ProjectExam
- Examiner2: Examiner
- Duration: int
- DurationPerStudentPerGroup: int
- BreakDurationPerGroup: int
- RedundantDuration: boolean
- RedundantDurPerStudentPerGroup: boolean
- RedundantBreakDurPerGroup: boolean
OralExam
- Examiner2: Examiner
- Duration: int
- DurationPerStudent: int
- BreakDurationPerStudent: int
- RedundantDuration: boolean
- RedundantDurPerStudent: boolean
- RedundantBreakDurPerStudent: boolean

Figure 3.1: Class Diagram

In the case of Oral and Project exams a problem has been identified with the core/redundant information, since they contain 3 attributes each that define the Duration of the given exam.

1. For the Project exam (Similar for Oral Exam) these being:
   (A) Duration
   (B) Duration per Student per Group
   (C) Break Duration per Group



Duartion	per Stud.	Break dur.
1	🟩 1	1
🟩 0	🟥 0	🟥 0
1	🟩 0	🟥 0
🟩 0	1	🟥 0
🟩 0	🟥 0	1
🟩 0	1	1
1	🟩 0	1
1	1	🟩 0

Where:
1 - inputted value by user
0 - missing value
🟩- value calculated out of the other 2
🟥- value set to default value

The correlation being: " $a = b * Students + c * Groups$ " where the third variable is calculated from the other two. This results in a necessary way to store the core information inputted by the user and determine what is redundant, achieved by the correlating **RedundantDuration**, **RedundantDurationPerStudentPerGroup**, **RedundantBreakDurationPerGroup** boolean attributes which define what is core information and what is not. If the user inputs 1 or 2 values out of 3, a default values for **DefultBreakDurationPerGroup** and/or **DefultDurationPerStudentPerGroup** is introduced, so the calculations can still be performed. These values are introduced in the **ExamPeriod** class since they are default and uniform across the whole examination period. If all 3 values are inputted, a prioritization chart shown in the Figure above is used to avoid problems with addition.

Furthermore, the lunch break also needs to be addressed, while avoiding possible "cuts" in the exam at random time. The **BrakeStartTime** and **BreakEndTime** attributes of the **ExamPeriod** are introduced which are responsible for handling cuts as shown in Figure 3.2

As shown in example 1 and example 3 in the same Figure, it ends/begins before the **BreakEndTime/BrakeStartTime** so there is no break needed in those particular cases. However, in the case of example 2, it starts before the **BreakStartTime** and ends after the **EndStartTime**, resulting in a brake automatically accounted for in an interval between 11:00-14:00(example values) avoiding interruptions of possible Oral or Project exams. It is important to mention that this behaviour is only applies to Oral and Project exams!

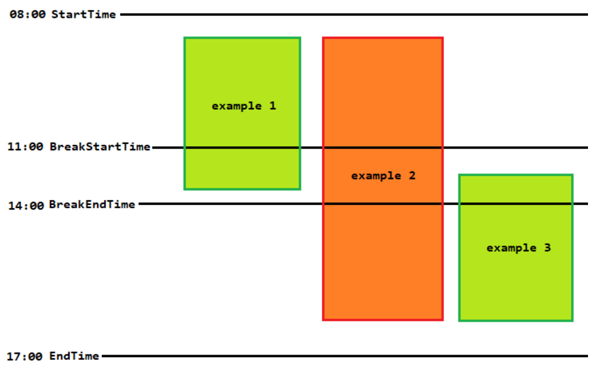


Figure 3.2: Lunch Brake Timing

Another behavioural pattern is defined by the **StartTime** and the **EndTime** attributes of the **Exam-Period**, which describe the time from when exams can take place and vice versa. When an exam ends after the **EndTime**, it gets cut of for that day and continues the following one from the **StartTime**.

The handling of the data, which includes the adding or removing of the class instances, has to be 100% reliable to avoid future data conflicts. The sequence diagrams in Figures 3.3 and 3.4 demonstrate the design of the **Examiner** class (other classes are based on the same principal).

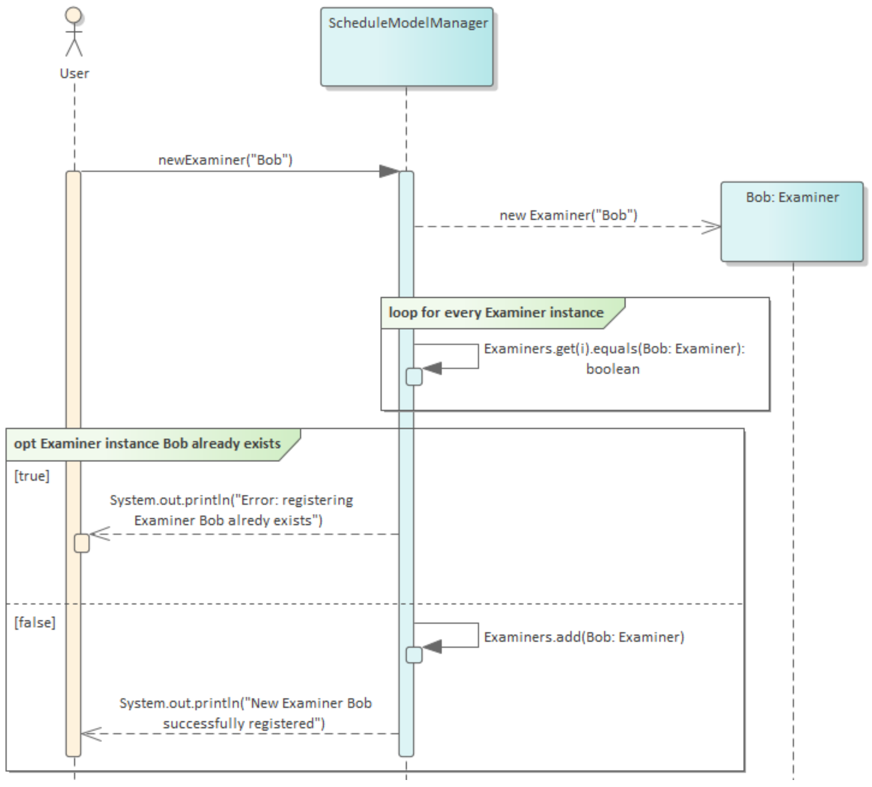

User
ScheduleModelManager
newExaminer("Bob")
new Examiner("Bob")
Bob: Examiner
loop for every Examiner instance
Examiners.get(i).equals(Bob: Examiner): boolean
opt Examiner instance Bob already exists
[true]
System.out.println("Error: registering Examiner Bob alredy exists")
[false]
Examiners.add(Bob: Examiner)
System.out.println("New Examiner Bob successfully registered")

Figure 3.3: Examiner sequence diagram 1

As shown in the diagrams, for each class there is a unique identifier (Name in each class except Student(VIAID)) that cannot match for two separate instances. Furthermore for complex classes, like

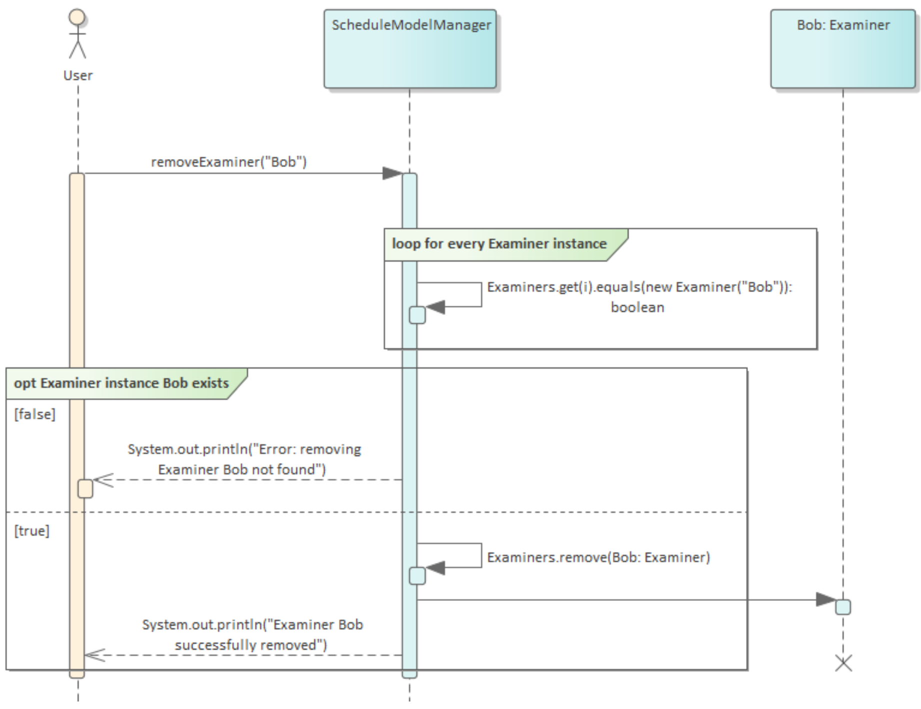

User
ScheduleModelManager
Bob: Examiner
removeExaminer("Bob")
loop for every Examiner instance
Examiners.get(i).equals(new Examiner("Bob")): boolean
opt Examiner instance Bob exists
[false]
System.out.println("Error: removing Examiner Bob not found")
[true]
Examiners.remove(Bob: Examiner)
System.out.println("Examiner Bob successfully removed")

Figure 3.4: Examiner sequence diagram 2

Exam, the design necessitates that they be dependant on the other classes' instances, thus not permitting construction without the existence of the instances associated with them.

When designing the UI, it is logical to have a dedicated, separate view for editing every class. This results in the need for a separate Controller for each of the classes as shown in Figure 3.5. (Lewis and Loftus, n.d.)

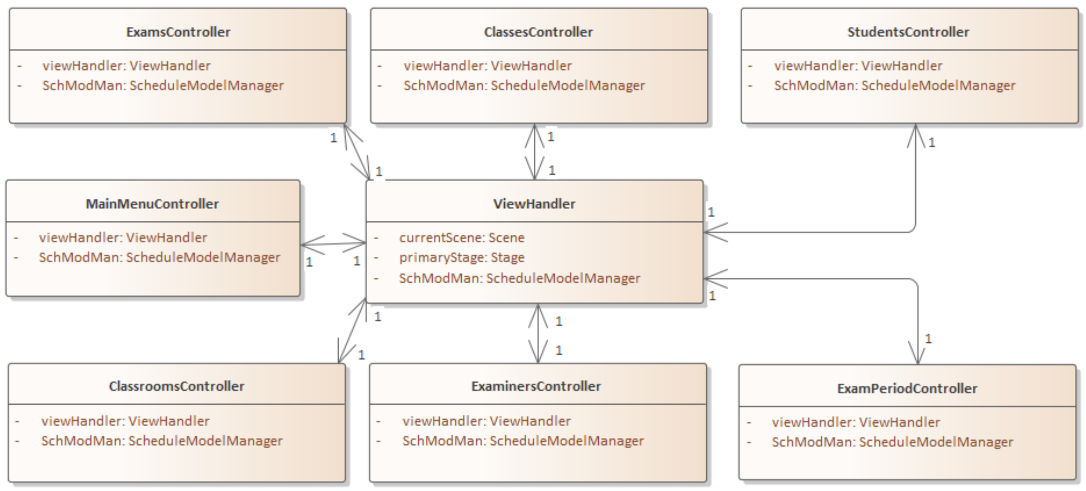

ExamsController
- viewHandler: ViewHandler
- SchModMan: ScheduleModelManager
ClassesController
- viewHandler: ViewHandler
- SchModMan: ScheduleModelManager
StudentsController
- viewHandler: ViewHandler
- SchModMan: ScheduleModelManager
MainMenuController
- viewHandler: ViewHandler
- SchModMan: ScheduleModelManager
ViewHandler
- currentScene: Scene
- primaryStage: Stage
- SchModMan: ScheduleModelManager
ClassroomsController
- viewHandler: ViewHandler
- SchModMan: ScheduleModelManager
ExaminersController
- viewHandler: ViewHandler
- SchModMan: ScheduleModelManager
ExamPeriodController
- viewHandler: ViewHandler
- SchModMan: ScheduleModelManager
1

Figure 3.5: ViewHandler Package

Taking into account all of the above mentioned design choices and also the requirements defined by the customer, it is clear that the layered architecture design pattern would fit perfectly for this project. It is the most popular pattern for small single user systems like the one in this project, while also being defined in Non-functional Requirement 15. Addressing the bottom (data) layer, as stated by the customer in Non-functional requirement 16th, the system has to use XML files. Moreover, to fulfil the 11th Functional requirement, concerning the webpage design, it was obvious that JavaScript is the optimal choice since it natively supports reading from an XML. It would be reasonable to use a separate file for each of the classes' instances (Students, Classes, Examiners, Rooms and Exams). To avoid the fact that JavaScript's(AJAX) visualization of the data has to import from multiple files, a 6th file is introduced where information only required for the website is stored. It is important that the system only writes to the file, and the webpage only reads from it. Taking everything into account the following layered architecture pattern was constructed as shown in Figure 3.6

As shown, the business and the persistence layers are opened and the webpage accesses the data layer directly.

The following Chapter: Implementation uses the information derived and described above to formulate the final software solution.

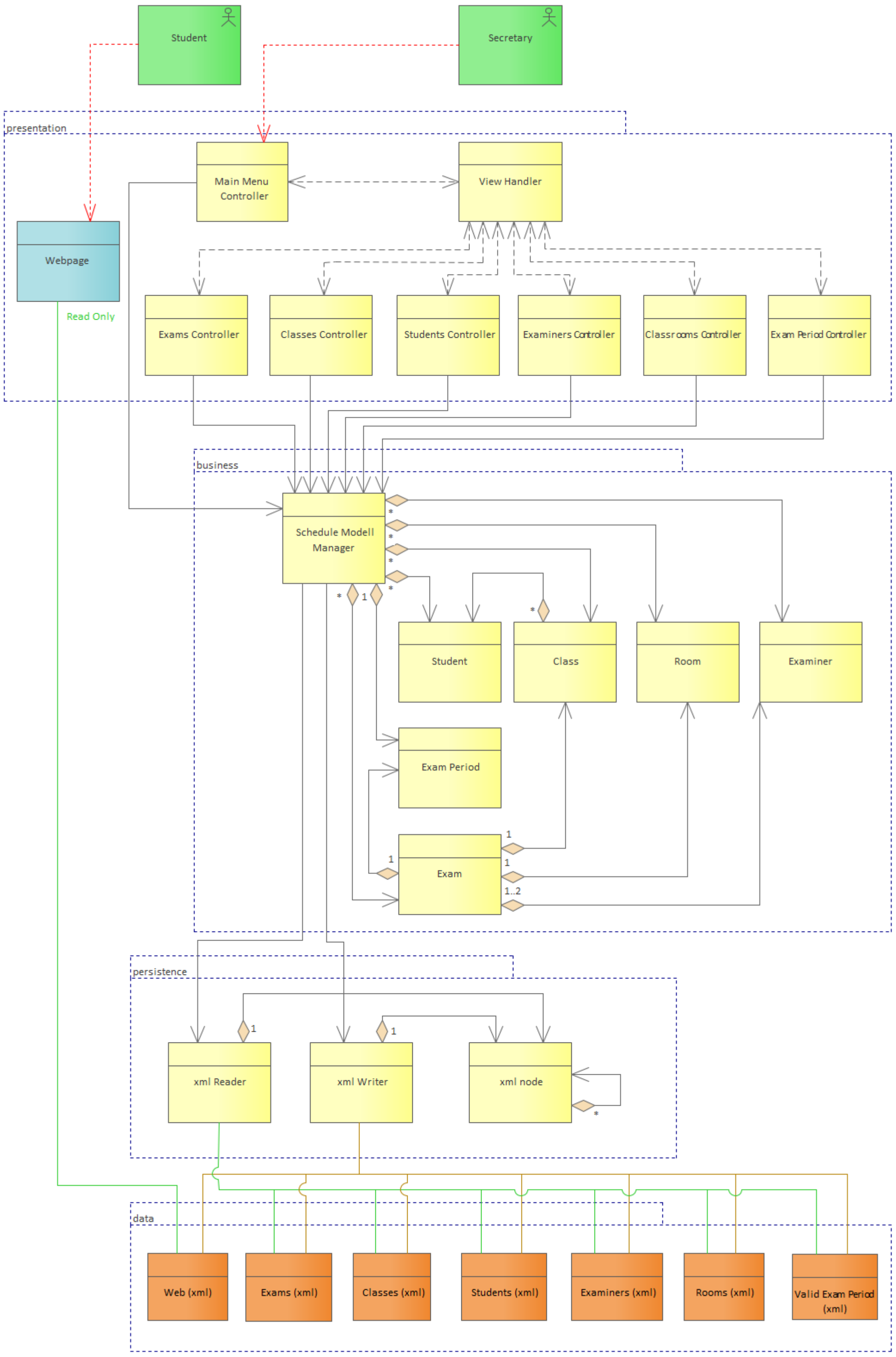
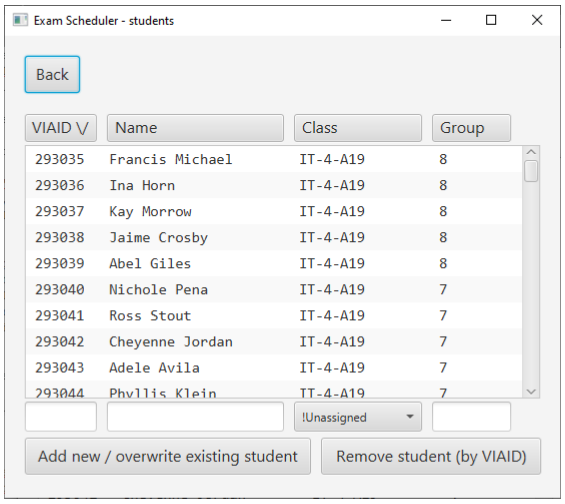


Figure 3.6: Layered Architecture Pattern

Chapter 4

Implementation

The implementation process was pretty straight forward after the class diagrams had been expanded with their operations (to inspect these, please refer to Appendix Diagrams). From the perspective of time, we implemented the business layer first, meaning the **ScheduleModelManager** package as a console application, that is why a lot of the executed operations are logged to the console (it was kept to make the testing easier), after that the persistence layer came, meaning the **xmlHandler** package.A choice was made not to use the java's DOM parser to handle xmls, instead we implemented our own **xmlHandler** package, which has some limitations but those do not affect the scope of this project. The decision was made because the implementation was simpler to create xml node trees and xml files from them compared to the DOM parser. Furthermore it follows the exact logic of JavaScript's DOMparser which was used in the web development process, thus achieving a nice uniform logic across both environments. The last part was the presentation layer with the view package.


Exam Scheduler - students
Back
!Unassigned
Add new / overwrite existing student
Remove student (by VIAID)

VIAID ⋁	Name	Class	Group
293035	Francis Michael	IT-4-A19	8
293036	Ina Horn	IT-4-A19	8
293037	Kay Morrow	IT-4-A19	8
293038	Jaime Crosby	IT-4-A19	8
293039	Abel Giles	IT-4-A19	8
293040	Nichole Pena	IT-4-A19	7
293041	Ross Stout	IT-4-A19	7
293042	Cheyenne Jordan	IT-4-A19	7
293043	Adele Avila	IT-4-A19	7
293044	Phyllis Klein	IT-4-A19	7

Figure 4.1: Presentation layer

Further on, this chapter will proceed in a logical order, describing the flow of information and operations from the top layer to the bottom layer.

It will start with an example of an add element use case, which is fairly similar to other elements from the perspective of implementation.

The UI shown on Figure 4.1 greets the user under the Students editing option (in this case some student are already added). The input system is foolproof in the sense that the corresponding input fields do not let the user type invalid information in the fields. In the cases where an element is depended on other elements such as in Participating Class, the user can only chose from a drop-down list ensuring that the dependent object exists.

The following snippet is the method concerned with the VIAID input field's checking (this is called every time a change is made to the field), where **ViaInp** is the input field itself:

```
public void Viaid()
  {
    if (!ViaInp.getText().equals(""))
    {
      CursorPos = ViaInp.getCaretPosition();
      ViaInp.setText(ViaInp.getText().replaceAll("[^\\d]", ""));
      if (ViaInp.getText().length() > 6)
        ViaInp.setText(ViaInp.getText().substring(0, 6));
      ViaInp.positionCaret(CursorPos);
    }
  }
```

As it is shown, first a test is made to check if the input field is not empty to ensure that it won't throw an exception after an attempt to make some operations on the content of the input field. Next the current caret position is saved to a variable, because if the content of the input-field is modified, the caret position will be reset. Following, using **RegEx** every non digit in the input field is replaced with empty strings and checked if the length of the string exceeds 6 characters. Ff it does, the excess is chopped and finally the caret position is set to its original position because of the automatic reset, as mentioned earlier.

This technique is pretty similar for all input fields using different **RegExes** and content checkers, but it always ensures that an exception is not thrown and makes invalid inputs impossible for the user.

As for the drop-down, it simply gets all of the instances of the Classes and setting them as options for the drop-down during the .init process, as shown below:

```
void init(ViewHandler viewHandler, ScheduleModelManager SchModMan)
```

```
  {
    this.viewHandler = viewHandler;
    this.SchModMan = SchModMan;
    ClsInp.getItems().add("!Unassigned");
    for (int i = 0; i < SchModMan.getClasses().length; i++)
    {
      ClsInp.getItems().add(SchModMan.getClasses()[i].getName());
    }
    ClsInp.getSelectionModel().select("!Unassigned");
    refreshList();
  }
```

Now that reliable inputs are inputted from the user, operations on them can be performed.

Further down only the more complex, interesting, and challenging operations of the code will be discussed, for any other detail, please refer to the JavaDoc or the source code itself. In addition any following mentions of **Date**refer to the system's Date class which means date and time in the conventional sense.

The most complex and sensitive method out of all is the **ScheduleModel.Exam.getEnd()** method, described as follows:

```
public Date getEnd()
  {
    Date end = Start.addMinutes(Duration);
    int dayStepper = -1;
    do
    {
      dayStepper++;
      if (!ValidExamPeriod.isValid(Start.addMinutes(dayStepper * 60 * 24)))
      {
        end = end.addMinutes(60 * 24);
      }
      else
      {
        if (Start.mergeDate(ValidExamPeriod.getBreakEndTime())
        .addMinutes(dayStepper * 60 * 24).isBefore(end) &&
            (Start.isBefore(Start.mergeDate(ValidExamPeriod.getBreakStartTime())
        .addMinutes(dayStepper * 60 * 24)) ||
            Start.equals(Start.mergeDate(ValidExamPeriod.getBreakStartTime())
        .addMinutes(dayStepper * 60 * 24))))
          end = end.addMinutes(ValidExamPeriod.getBreakDuration()); \\
        if (Start.mergeDate(ValidExamPeriod.getEndTime())
        .addMinutes(dayStepper * 60 * 24).isBefore(end))
          end = end.addMinutes((24 -
              ValidExamPeriod.getEndTime().getHour() +
              ValidExamPeriod.getStartTime().getHour()) * 60 -
              ValidExamPeriod.getEndTime().getMinute() +
              ValidExamPeriod.getStartTime().getMinute());
      }
    }
    while (dayStepper != end.toDays() - Start.toDays() && dayStepper < 20);
    if (dayStepper == 20)
      end = new Date(1,1,1,0,0);
    return end;
  }
```

The behaviour of Exams and Breaks have been detailed in the Design Chapter,as this method follows that logic. First the Duration gets added to the **StartDate** and the **sayStepper** gets set to -1, but it is only -1 to compensate that when the do-while loop begins it gets added up to 0. The whole problem here is that while scenarios are checked for the postcondition of increasing the **endDate** of the Exam, the checking process relies on using the **endDate**, thus the testing conditions are dynamically changing mid testing. This is handled by the do-while loop's condition which stops when the **dayStepper** reaches the day-difference between the **endDate** and the **StartDate**, making the loop run until the checking process finished running every day from StartDate to endDate, checking one day at the time, treating the testing condition dynamically. Inside the loop testing is performed if the endDate is valid, if not, then a step a day ahead is made and 24 hours added to the endDate. If it is valid, we test if a lunch break has to be inserted(based on the previously described behaviour in the Design), if positive, we add the **BreakDuration** to the /textbfendDate. Finally it tests if the exam's end is after the EndTime, and if it is, it adds the time to the endDate that is between the StartTime and the EndTime, basically continuing the Exam on the next day.

The method, however, has a fatal drawback that needs to be handled, that is , when the endDate is out of the ValidDates, it gets stuck in an infinite loop, since the day difference between the StartDate and endDate does not stop increasing. This exception is handled with a hard-coded exit condition after the maximum of 20 loop cycles, thus meaning that we assumed no exam will be longer than 20 days. And in the case of a 20 loop cycle run, the endDate gets set to 0001.01.01 00:00 indicating that the Exam's endDate is out of the scope of the ValidDates.

For the Ordering feature (which is visible in the top section of the window) the software uses bubble sorting since its the most simple one, and for the size of this project, the "slowness" of it does not matter. The code follows the commercial bubble sorting algorithm:

```
while (swapped)
        {
          swapped = false;
          for (int i = 0; i < Students.size() - 1; i++)
          {
            if ((Students.get(i).getName().compareTo(Students.get(i +
                1).getName()) > 0 && !reverse) ||
                (Students.get(i).getName().compareTo(Students.get(i +
                1).getName()) < 0 && reverse))
            {
              Students.add(i, Students.get(i + 1));
              Students.remove(i + 2);
              swapped = true;
            }
          }
        }
```

The persistence layer's exciting part is the raw xml's mapping process into a tree of xml nodes.

```
public node(String Name, ArrayList<String> Content)
  {
    this.Name = Name;
    this.Content = Content;
    this.isTextNode = false;
    Children = new ArrayList<node>();
    String tag;
    for (int i = 0; i < Content.size(); i++)
    {
      tag = "";
      if (Content.get(i).contains("<") && Content.get(i).contains(">") &&
         !Content.get(i).contains("</") && !Content.get(i).contains("<!"))
      {
        tag = Content.get(i).substring(Content.get(i).indexOf("<") + 1,
            Content.get(i).indexOf(">"));
        for (int j = i + 1; j < Content.size(); j++)
        {
          if (Content.get(j).contains("</" + tag + ">"))
          {
            Children.add(new node(tag,
                new ArrayList<String>(Content.subList(i + 1, j))));
            i = j;
            break;
          }
        }
      }
      if (Content.get(i).split("<").length == 3 &&
         Content.get(i).contains("</"))
      {
        tag = Content.get(i).substring(Content.get(i).indexOf("<") + 1,
            Content.get(i).indexOf(">"));
        Children.add(new node(tag, Content.get(i)
            .substring(Content.get(i).indexOf(">") + 1,
               Content.get(i).indexOf("</"))));
      }
    }
  }
```

After the FileReader read the xml file and constructs the root node of it, the root node will map its own direct children and construct them. After that the children will map of their children and so on, until everything is mapped.

Chapter 5

Testing

Following the requirements stated in the Analysis, originating the instructions from the Main Menu:

1. As a secretary I would like to create an exam schedule.
   - Click on "Clear schedule"
   - requirement fulfilled
2. As a secretary I would like to add elements to the exam schedule.
   - Click on "Exams" or "Classes" or "Students" or "Examiners" or "Classrooms"
   - Input the required data in the input-fields and select them in the drop-downs
   - Click on the corresponding add button
   - requirement fulfilled
3. As a secretary I would like to save the exam schedule.
   - Click on "Save schedule"
   - Select the save path and input the file name in the pop up window
   - Click save
   - requirement fulfilled
4. As a student or examiner, I would like to be able to access the current schedule
   - Load or create a schedule which has data to visualize
   - Click on "Update webpage"
   - Set up a web server in the USER.HOME/.schedule/web/ folder, hosting the prototype of our system

- Open it in any browser as localhost:8080 (we specified this port for testing but it can be something else)
- Go to the schedule tab on top
- Click on the desired class to see the exam period's weeks
- Click on the desired week to see the schedule for the selected class and week
- requirement fulfilled

5. As a secretary I would like to access previously saved exam schedules
    - Click on "Load schedule"
    - Select the previously saved schedule file (.sch)
    - Click open
    - requirement fulfilled
6. As a secretary I would like to remove elements from the exam schedule
    - Click on "Exams" or "Classes" or "Students" or "Examiners" or "Classrooms"
    - Input the unique id of the element which is needed to be removed, or click on the element in the list
    - Click on the corresponding remove button
    - requirement fulfilled
7. As a secretary I would like to edit elements in the exam schedule
    - Click on "Exams" or "Classes" or "Students" or "Examiners" or "Classrooms"
    - Input the unique id of the element which is needed to be edited, or click on the element in the list
    - Input the data that requires change (unique id can not be changed, in that case the element needs removing and then re-adding (the classes are exceptions from this))
    - click on the corresponding add/overwrite button
    - requirement fulfilled
8. As a secretary I would like to check for conflicts in the exam schedule
    - Click on "Exams"
    - Click on "Check conflicts"

- requirement fulfilled (Details have to be expanded to see the result)

9. As a secretary I would like to edit the attributes of the exam period
    - Click on "Exam Period"
    - input the required data
    - requirement fulfilled
10. As a secretary I would like to load previously saved exam schedules
    - Click on "Load schedule"
    - Select the previously saved schedule file (.sch)
    - Click open
    - requirement fulfilled
11. As a student I would like to get a visual representation of the exam schedules
    - Load or create a schedule which has data to visualize
    - Click on "Update webpage"
    - Set up a web server in the USER.HOME/.schedule/web/ folder, hosting the prototype of our system
    - Open it in any browser as localhost:8080 (we specified this port for testing but it can be something else)
    - Go to the schedule tab on top
    - Click on the desired class to see the exam period's weeks
    - Click on the desired week to see the schedule for the selected class and week
    - requirement fulfilled
12. As a student I would like to see my detailed exam schedule for a specific oral or project exam
    - Not implemented yet
13. As a secretary I would like to get an automatically generated schedule for every project and oral exam
    - Not implemented yet

Chapter 6

Results and Discussion

The conclusions on how to interpret the results on the developed system were only formulated when the testing phase was complete. It is necessary to discuss our initial expectations and how they compare to the resulting outcome.

Our vision for the system included a sturdy design that minimizes error when processing information whilst having a user friendly graphical interface and appealing visual representation on a website. It is of course, reasonable to maintain realistic expectations while designing and developing a solution as no final product can match our personal expectations and those of the customer. That is why, a "reality check" had to be initiated multiple times throughout the duration of this project to keep group members informed on what is reality and what is fiction.

The technologies, and methods we wanted to use often proved to be outside of the scope of this project. At certain areas like design the current level of knowledge was insufficient to satisfy our expectations, resulting in downgrading of what was planned and what is achieved. That is not to say, disappointing, but rather unsatisfactory compared to our initial vision for the solution.

The achieved results were sufficient. The guidelines set for the current semester were followed as closely as possible with minimal deviation. The Project description was constructed first, followed by Analysis inheriting the essence of the background description and problem formulation. After defining the Requirements, we began brainstorming possible design and architecture solutions that match the written Analysis. After structuring the domain objects into a Domain Model and extending them to a Class Diagram, the design structure was solidifying. As shown in Chapter 3, we took the liberty to discuss the more important design choices such as time management and class handling while also providing information on the design choices of the user interface and how data should be displayed in the website. The above process resulted into a steady implementation routine closely mimicking the set design choices while permitting the liberty for implementing extra features that could build on top of the design. The final results did not meet our initial expectations, but followed the routine set by the VIA guidelines.

Chapter 7

Conclusion

Finalizing the Chapters and comparing our results to the Requirements set by the client, allows us to reflect and conclude this report .

First, it is important to reflect on the work conducted is this project, and look back in chronological order.

We began by writing a project description, which served as a guiding material for further work. It included a background description, problem formulation, and delimitations that formulated our understanding on the scope of the project.

The Project description was followed by the Analysis part, which involves analysing the domain of the problem and setting up Requirements based on the customer's criteria. The Requirements were compared to our solution in Chapter 5: Testing and the results were adequate enough to consider the implementation a moderate success.

The Analysis chapter, logically led to the Design, which used the artefacts of the analysis to create a model and define the architecture of the system.This allowed us to solidify our idea of the implementation and set the basic building blocks for a gradual implementation.

The implementation was a collection of all previous information assembled into a coherent whole, resulting in a software solution closely resembling our initial expectations for the system.

The final outcome was that we developed a system, following an architecture we designed, combined with a satisfactory Graphic Interface for data input. The last part was a custom made website, integrating the XML files generated by the Java application and displaying them using AJAX in a dedicated Schedule Tab on the website.

In conclusion, we have closely followed our initially set criteria using a linear approach defined by the

Methodology for this semester, to achieve a software solution that is hopefully sufficient in following the customers expectations.

Chapter 8

References

En.via.dk. 2019. About VIA. [online] Available at: <https://en.via.dk/about-via> [Accessed 20 December 2019].

Eide, P., 2004. 20. [online] Idi.ntnu.no. Available at: <https://www.idi.ntnu.no/grupper/su/fordypningsprosjekt-2005/eide-fordyp05.pdf> [Accessed 20 December 2019].

Lewis, J. and Loftus, W., n.d. Java Software Solutions. 7th ed. Pearson, p.350.

Appendix A

Process Report

A.0.1 Introduction

Throughout the first semester of VIA Software Engineering we have worked on the semester project and four courses: SDJ, RWD, SSE, MSE. The courses have presented some challenges that we have tried to solve both individually and in the group environment.

The case for the semester project is Exam Scheduling System which purpose is to efficiently schedule and distribute exam sessions and students. The project is heavily influenced by the knowledge gained in SDJ, RWD and SSE while some elements of MSE are helpful for the general purpose of developing more efficient systems which requires mathematical background.

SDJ is the most used course throughout the development of the project. The system is built entirely in Java which means that regular attendance and dedication to exercises and homework was crucial to obtaining the necessary knowledge in order to create the system.

RWD is the second crucial course for the development of the system. A responsive website is a requirement to display the exam information in a user friendly manner, therefore completing the obligatory assignments was a necessary step to further the project development. The appropriate amount of hours were dedicated to creating and designing the responsive website.

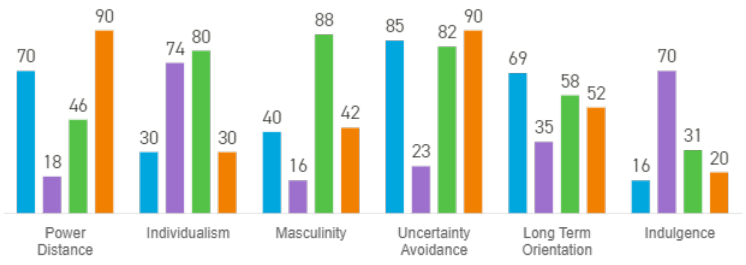
SSE is the course that allowed us to get insight on how to work better in the group environment and presented the guidelines necessary to follow VIA documentation writing models.

Although MSE did not provide much technical relevance to the project, it was helpful in providing mathematical background for writing better code based on mathematical concepts which later can be applied to algorithms.

Our group meetings are regular and weekly, containing at least one discussion session and one work session. The time intervals vary depending on the purpose of the meeting and the amount of work scheduled to be done.

A.0.2 Group Description

Working for the first time in an international environment, the first thing that we noticed were the cultural differences between us. Our group is composed of members from different countries, thus different cultures and educational backgrounds (Philip Philev from Bulgaria, Levente Nagy from Hungary, Roberto Fat and Tiberiu Marian from Romania). What helped bridge the cultural gaps between us were mainly the Hofstede insights, as presented below:



1. Bulgaria Blue;
2. Denmark Purple;
3. Hungary Green
4. Romania Orange

As it can be seen, while our countries are quite similar in certain aspects, they are very different in others. What was obvious to us from the onset was the significant difference in the power distance of our work environment (Denmark) compared to our home countries. Nevertheless, we quickly and easily adapted to this, and as a result our decision-making process became very democratic, with no one person asserting dominance over the others. This in turn contributed to a healthy and, most importantly, productive group environment. Despite the values shown on the graph, the educational systems of our countries are very individualistic in contrast to Denmark, where the educational system is much more collective and focused on group work. While we were all used to individual school work, some of us had never taken part in a group project of this magnitude and, naturally, adapting to this situation was challenging. Working in a group environment allows us to explore our individual strengths while others can compensate for our weaknesses.

As part of our SSE classes, we were given the opportunity to complete an E-estimate Personal Profile in order to better comprehend the individual traits of each member. As it turned out, each member possesses unique characteristics in the scope of the group, as shown below:

1. Levente predominantly red, followed by even yellow and blue with little green
2. Philip predominantly yellow, followed by even blue and green with little red
3. Roberto predominantly blue, followed by green, little yellow and very little red
4. Tiberiu predominantly green, followed by almost equal blue with little yellow and red

A.0.3 Project Initiation

Initially the client presented us with a set of problems that should be solved as good as possible based on our current knowledge by an Exam Scheduling System. Those problems are based on feedback from the VIA secretary. After receiving the information we were presented with the challenge how to initiate the problem as members worked on different speeds and that posed a challenge to the general cohesion of the group. We decided to continue as first semester optimism often prevails. We agreed on the set of tools to use in order to achieve maximum results with the "hand we were dealt". The following weeks were basic introduction to the structure and methods used in the group.

A.0.4 Project description

Writing the project description was not too challenging, nevertheless a correct way to approach the problem had to be made. Some group members were in separate groups having different degrees of understanding of the project description, resulting in a varied definition on how to handle the problem. Some members were not keen on further reflection and wanted to begin working on the already given topic. This posed a challenge since, a good understanding of the building blocks had to be made, an some arguments arisen. As different group members had performed differently on the project description presentation it was important to normalize how we collectively saw the problem.

A.0.5 Project Execution

The main phase of the project began when the analysis and the requirements were set. Then we could start working on figuring the design behind the actual implementation. First the tools to be used for writing, version control and management. Those tools include

1. GitHub A version control software allowing for remote synchronization of writing, code and other documentation
2. LaTeX A TeX editor offering wider functionality and freedom for documentation compared to Word
3. Microsoft Teams Software for Remotely handling and working on documents

Having determined the tools to be used, research and analysis began and each group member had to research how to design aspects of the system.

A.0.6 Personal Reflections

Philip Philev

I tried to closely follow the guidelines set by VIA for both documentation and project development, thus I am moderately satisfied with the amount of work I dedicated in order to develop and document the system. Sadly the group contract did not serve its purpose. I see that I am also at fault for not insisting on following the set rules more closely. This will change in the upcoming semesters. The contract will be followed more precise and negative consequences will be enforced. I feel responsible in both a positive and a negative way. I could have expressed my intentions better, which could have resulted in better group work altogether Unfortunately the work distribution was subpar. Not everyone contributed the necessary amount, or expressed sufficient interest in development or/and documentation, resulting in increased workload and last minute gap filling due to insufficient provided materials. As frustrating as this may be, it was a valuable lesson for future collaborations and group dynamics.

Levente Nagy

For our group member choices, we teamed up with people with similar interest, intellect and humor, so basically the classmates who we feel comfortable around more like as a friend than a college. This made the group work fun, when it was not efficient. As for the group I think we set the goal and our expectations much higher than what the âĂŸnormalâĂŹ was, but still on the edge of the achievable level. But as we got to the project weeks before the winter break, things just fallen apart. According to the group work done, before these 2 weeks, we should have been able to work very well as a team and as one unit. Thus, as we were having a democratic group model, no one led the team and took control, which would have been a solution to our problem, or the other way that we should have done, and definitely will do next semester is writing a very strict group contract and actually enforcing it to everyone, also very strictly. But we learned from it, and hopefully will correct our mistakes next semester.

A.0.7 Supervision

Supervisors were not used to their maximum potential. This was instructive and informative as towards the end we realised we could have used the provided resources much more efficiently. Nevertheless supervisors were of great help when we had the opportunity to discuss aspects of the design and implementation paving the way to some of the choices we made in this project report.

A.0.8 Conclusion

In conclusion, the group environment was good and very friendly. Nevertheless, we could not collaborate efficiently, which lead to drawbacks in the documentation process. The courses throughout the semester were helpful in their respective element, offering guidance on how to approach the engineering problems we were facing. Positive things about the group work:

1. Friendly environment
2. Common areas of interest
3. Interesting problem to tackle
4. Multiple resources and opinions to digest

Negative things about the group work:

1. Lack of enthusiasm at critical points
2. Insufficient collaboration
3. Communication issues
4. Skill gap

List of recommendation

1. More expressive on roles in the group
2. Understanding the concept of groupwork
3. Dedication to the common goal