AAMS

(Automatic Assignment Management System)

Group 21

Ajanth Thangavelu

Roberto Castañeda Lozano

Tony Karlsson

Love Jädergård

Design document

Contents

List of Figures

1. Introduction

Purpose

This document describes the design of AAMS and provides the intended audience information needed in order to the build system.

Scope

After reading this document:

- A project manager shall be able create a detailed implementation plan and estimate time and cost of the remaining development.
- A developing team should be able to write code based on the information provided.

Intended Audience

- Developers
- Project managers
- Universities and other institutions interested in further developing of AAMS.

Version

AAMS version 1.0

References

- Requirements Document (see last page of the document)
- RES system administration Documents
- *wxWidgets* (`http://www.wxwidgets.org/`)
- *TinyXML* (`http://www.grinninglizard.com/tinyxml/`)
- *VMime* (`http://www.vmime.org`)
- *Haru Free PDF Library* (`http://libharu.sourceforge.net/`)

Important terms

API: Application Programming Interface, source code interface that an operating system or library provides to support requests for services to be made of it by computer programs.

assignment: course work with a defined deadline. The system assumes that the assignment must be handed in electronic format, corresponding to a single student.

attachment: in the system context, file that is included in an outgoing e-mail.

e-mail account: in the system context, e-mail address that identifies a location to which e-mail messages can be delivered, usually provided to the user by a company or institution (e.g. the university of the teacher).

C++: general-purpose, imperative programming language, regarded as a mid-level language, as it comprises a combination of both high-level and low-level language features.

course: in the system context, a set of assignments related to some students and a professor.

e-mail client: In the system context, front-end computer program used to manage e-mail.

e-mail server: computer running a program that transfers electronic mail messages from one computer to another.

e-mail (electronic mail): method of composing, sending, storing, and receiving messages over electronic communication systems. In the context of this document we will always refer to it as the Internet based system.

subject line: field of the header of an e-mail that contains a brief summary of the contents of the message.

FreeBSD 6.x: Unix-like free operating system descended from UNIX via the Berkeley Software Distribution.

feedback: in the system context, information about an assignment that is given back from the professor to the student.

GNU/Linux Ubuntu 7.x: predominantly desktop-oriented Linux distribution based on Debian GNU/Linux.

GPL: General Public License, widely used free software license, originally written for the GNU project. It is a strong copyleft license that requires derived works to be available under the same copyleft.

GUI: Graphical User Interface, type of user interface which allows people to interact with a computer presenting graphical icons, visual indicators or special graphical elements called "widgets", instead of offering only text menus, or requiring typed commands.

grade: in the system context, teacher's standardized evaluation of a student's work.

header: section of an e-mail that consists of fields such as summary, sender, receiver, subject, and other information about it.

homework: see **assignment**.

Integer: In the system context, data type that represents some finite subset of the mathematical integers.

Internet: worldwide, publicly accessible series of interconnected computer networks that transmit data by packet switching using the standard Internet Protocol.

mailbox: In the system context, folder where messages are delivered and stored, e-mail equivalent of a Letter box.

mark: see **grade**.

POP3: Post Office Protocol version 3, Internet standard protocol used to retrieve e-mail from a remote server.

plain text: ordinary "unformatted" sequential file readable as textual material without format processing.

professor: in the system context, the person in charge of collecting, commenting and grading the assignments from a course.

RES system: system used in the CSC (Computer Science and Communications) department from the KTH (Kungliga Tekniska Högskolan) university for the storage and management of courses information (grades, attending students...).

SMTP: Simple Mail Transfer Protocol, Internet standard for e-mail transmissions across the Internet.

Secure Shell: Secure Shell, network protocol that allows data to be exchanged over a secure channel between two computers.

String: In the system context, data type that represents an ordered sequence of characters.

student: in the system context, the person that must elaborate and send the assignments of the course to the professor.

teacher: see **professor**.

UTF-8: 8-bit UCS/Unicode Transformation Format, variable-length character encoding for Unicode, an industry standard allowing computers to consistently represent and manipulate text expressed in most of the world's writing systems.

user: see **professor**.

Windows XP: Windows eXPerience, line of operating systems developed by Microsoft for use on general-purpose computer systems, including home and business desktops, notebook computers, and media centers.

wxWidgets: widget toolkit for creating graphical user interfaces (GUIs) for cross-platform applications, enabling a program's GUI code to compile and run on several computer platforms with minimal or no code changes.

XML: eXtensible Markup Language, general-purpose markup language that provides a way to combine a text and extra information about it.

Abstract

This document provides detailed data of AAMS:

- Architecture and design issues
- Graphical User Interface
- Components and their relationships with each other
- Classes and their methods
- Functionality test cases to verify the planned implementation

2. System overview

2.1. General description

As it was described in the Requirements Document ([2]), the functionality of AAMS can be summarized as follows:

The system will show a graphical interface through which the user will be able to perform these general tasks:

- Define a course with assignments and students for it.
- Interact with the e-mail server, in order to collect assignments, or collect them manually.
- Organize the assignments to every student and course.
- Attach comments and marks to each assignment and send them back to each student.
- Create reports in order to make a general vision of the state of the course and relevant information.
- Communicate with the RES system (used to register students in courses, add marks to their assignments, etc.) and link other external system communication plug-ins.

Internally, the system will follow the typical structure for a classic event-driven application that manages information from a central repository. There will be two main components: the component in charge of storing and serving the data to all the surrounding components (kernel in the application data-flow), and the component in charge of catching all the user events and coordinating the actions triggered by these (top in the hierarchy in the application control-flow). These two main components and the rest will be described in further detail in the next sections of the Design Document.

2.2. Overall architecture description

The system will follow the repository model discussed in [1], where the shared data will be stored in files (figure 2). As the system will include a User Graphical Interface, the control style will be the event-driven model, which is also discussed in the course book: all user actions will be handled by the Event handler, which will process these control requests back and forth between all four communication component (figure 1). The Event handler is implemented with the help of the framework *wxWidgets*.

The Data storage component will be used as the main gate to retrieve and send data to every component (see figure 2), except the RES communication which is a subcomponent depending on data from the External communication component. The data itself will be saved in three XML files.

2.3. Detailed architecture

A brief description of the architecture that the system will follow, by the explanation of each component of it, will be given in this subsection. For every component we will show a relaxed

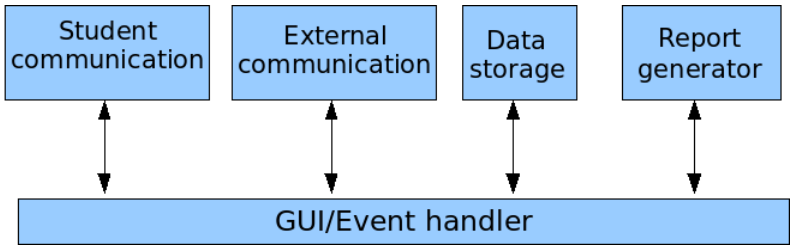


Figure 1: Control-flow model of the whole system

version of the standard data-flow diagrams, where boxes with labels represent subcomponents and arrows between them represent data flowing in the specified direction. Control-flow diagrams will be also shown. In these, the arrows represent the control relationship between subcomponents (if there is an arrow from A to B, then A has the ability of deciding when B should be required). All the subcomponents that form a whole component will be represented inside an slashed polygon, and external components to the component that is being represented in each diagram will be represented by just their names between parenthesis.

Student communication component

The Student communication component will be implemented as two layers: the e-mail library abstraction layer and the AAMS e-mail functionality layer. The e-mail library abstraction layer hides everything library specific from the rest of the system. This will also have the effect that the e-mail library can be replaced with another library and only the e-mail library abstraction layer will have to be changed. The AAMS e-mail functionality layer is the part of the student communication component that interacts with the rest of the system. Any activity from the student communication component is initiated by the GUI. The student communication component will control the data storage component to send or receive data.

External communication and RES communication components

The intention of the external communication component is to provide some (limited) synchronization capabilities with other systems that are usually implemented in every institution where our system could work. Therefore, the system will be able to import the list of students for a course and to export the grades of these through an intermediate representation. A layered structure (consisting of an abstract common layer and the specific one for every external system) has been designed in order to make the system more flexible to new possible ports for different specific systems, like the RES system.

The External communication component will be decomposed in two basic subcomponents:

Export grades

This subcomponent will be in charge of creating an intermediate representation file, from the course file, with information about students and grades in one specific assignment that can be after used to feed an external system through its specific component. It is controlled by the GUI/Event handler component.

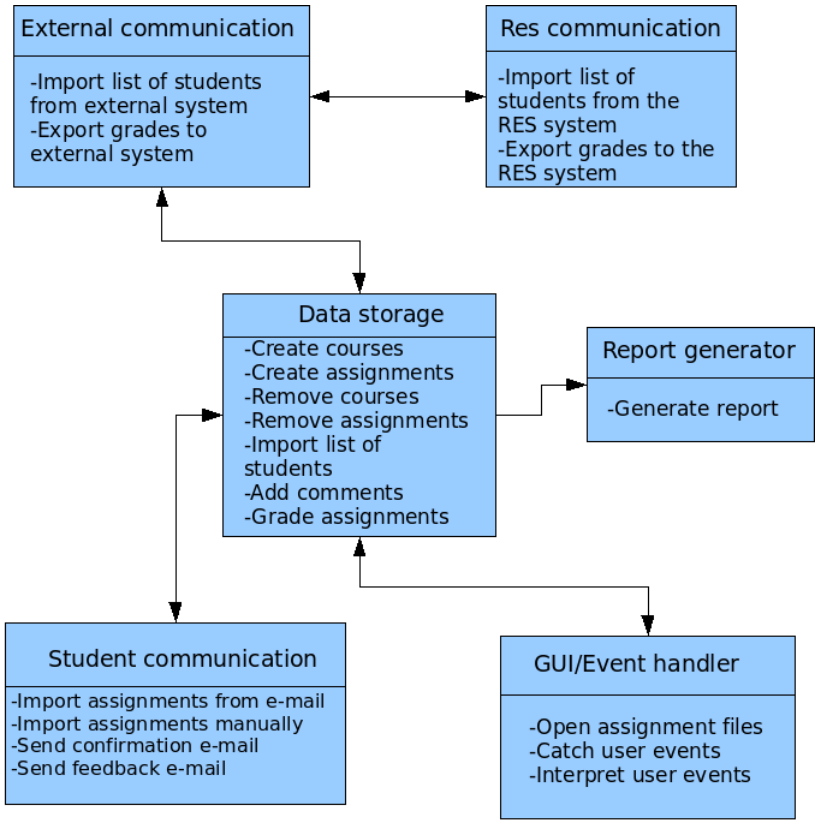


Figure 2: Data-flow model of the whole system

Import list of students
This subcomponent will be in charge of entering in the course file a list of students, with their basic information, from an intermediate representation file that has been obtained from an specific external system, through its import component. It is controlled by the GUI/Event handler component.

The RES communication component will consist of specializations of the two listed subcomponents:

Export grades to RES system
This subcomponent will be in charge of taking an intermediate representation file created by the Export grades subcomponent and using it to enter the grades of an specific assignment in the RES system. It is controlled by the generic Export grades subcomponent.

Import list of students from RES system
This subcomponent will be in charge of creating an intermediate representation file, from the RES system, with a list of students that belong to an specific course. This file will be used by the generic Import list of students subcomponent in order to enter them in a course file. This subcomponent is controlled by the generic Import list of students subcomponent.

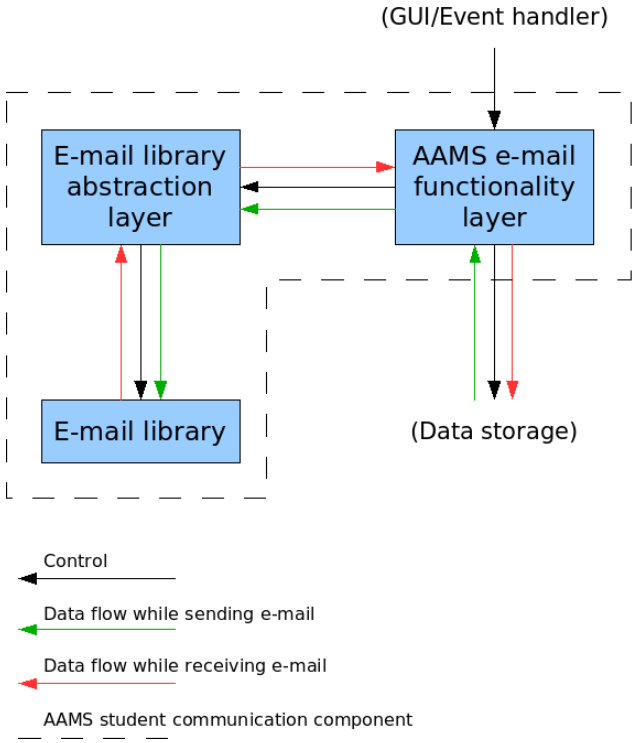


Figure 3: Control and data-flow model of the student communication component

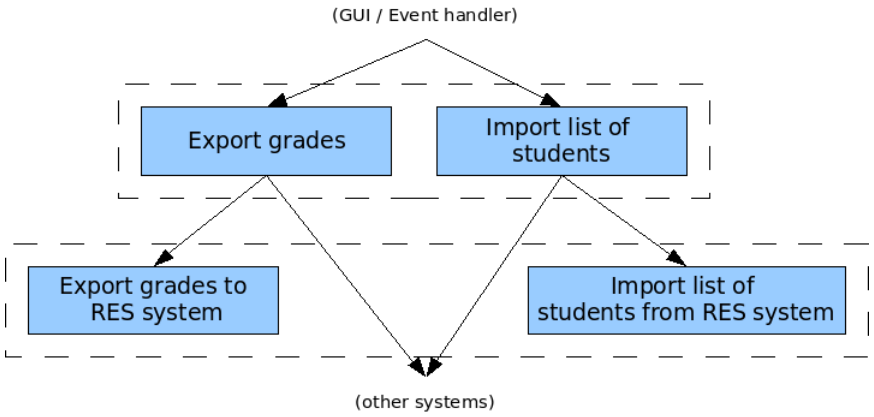


Figure 4: Control-flow model of the external communication and RES communication components

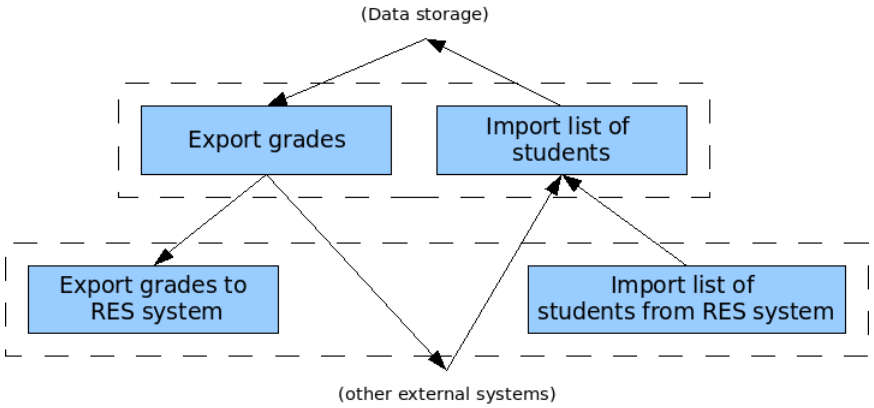


Figure 5: Data-flow model of the external communication and RES communication components

GUI/Event handler component

The Graphical User Interface will be built with the help of the framework *wxWidgets*, a cross-platform object-oriented GUI library that provides different tools for an extensive number of ports. The system will follow the classical event-driven scheme, as it is shown in the overall architecture description: the application starts by building the starting GUI (GUI creator component), sits in a loop waiting for events initiated by the user, which are caught by the event catcher component and managed by the GUI controller.

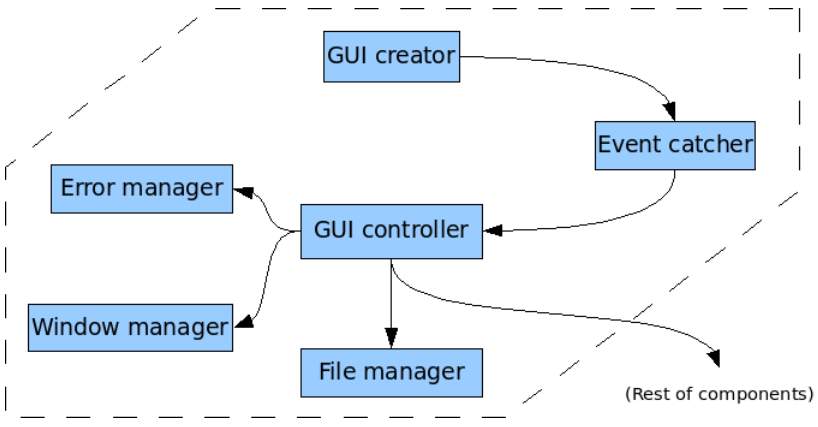


Figure 6: Control-flow model of the GUI/Event handler component

The GUI/Event handler component will be decomposed in these subcomponents:

GUI creator

This subcomponent will be in charge of defining and drawing, through the *wxWidgets* framework, the starting Graphical User Interface. Internally, the data generated in this subcomponent (information about the GUI structure) flows internally the GUI library towards the Event catcher subcomponent. This subcomponent is in the top of the control hierarchy, because it is required at the starting point of the system.

Event catcher

This subcomponent will be almost entirely provided by the *wxWidgets* framework. It will be in charge of catching and classifying the user events (and other possible events generated by the own system or external systems), and sending data (information attached to every event) to the GUI controller subcomponent. This subcomponent is controlled by the *wxWidgets* library, through the GUI creator subcomponent.

GUI controller

This subcomponent will be in charge of handling all the possible events that can be caught by the Event catcher subcomponent, controlling and sending data to the rest of components of the system. It is controlled by the Event catcher subcomponent.

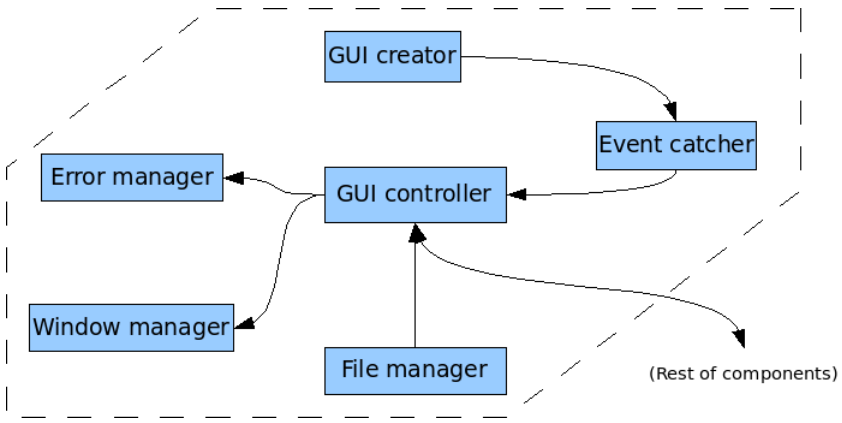


Figure 7: Data-flow model of the GUI/Event handler component

Error manager

This subcomponent will be in charge of managing (warning the user, saving information, etc.) all possible errors that can be detected by the GUI controller subcomponent during the execution of the system. It is controlled by the GUI controller subcomponent.

Window manager

This subcomponent will be in charge of creating and managing the secondary windows or forms that will be shown during the execution of the system (report forms, etc.). It is controlled by the GUI controller subcomponent.

File manager
This subcomponent will be in charge of handling the operative system level tasks of opening, creating and closing files that will be possibly processed later by other components of the system (mainly the Data storage component). It is controlled by the GUI controller subcomponent.

Data storage component

The data storage component handles all the data stored by the system. The data storage component is divided into three parts, each handling the storage of the data associated with it: “Course

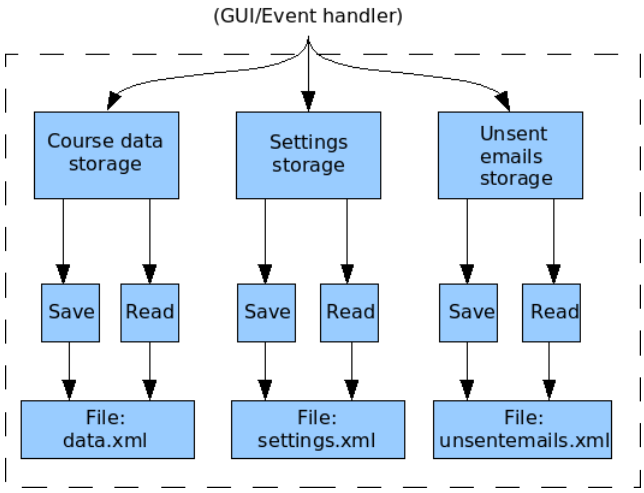


Figure 8: Control-flow model of the data storage component

data storage" part handles the storing of data from the "data.xml" file. "Settings storage" part handles the storage of data from the "settings.xml" file. "Unsent emails storage" part handles the storage of data from the "unsentemails.xml" file. Each part have a save and a read task that saves and reads from their files.

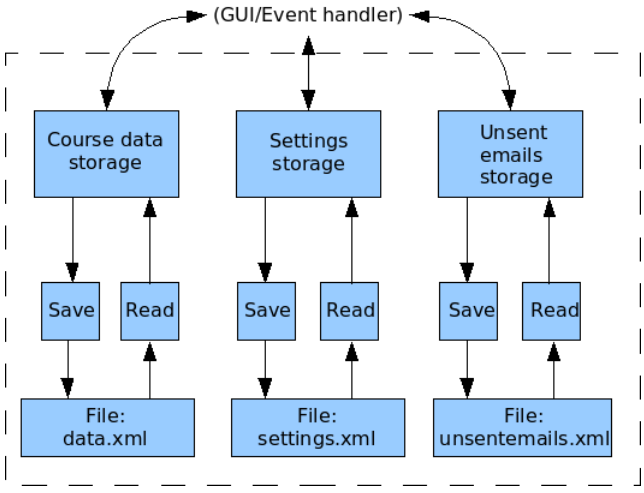


Figure 9: Data-flow model of the data storage component

Report generator component

The main and only purpose of this component is to generate reports of students, assignments and courses. the Report generator component will be executed from an user action which is handled by the Event Handler/GUI and thereafter retrieval of data from the Data storage is allowed. Generation of the reports is made in an internal component (Generate), depending on which type of report is requested.

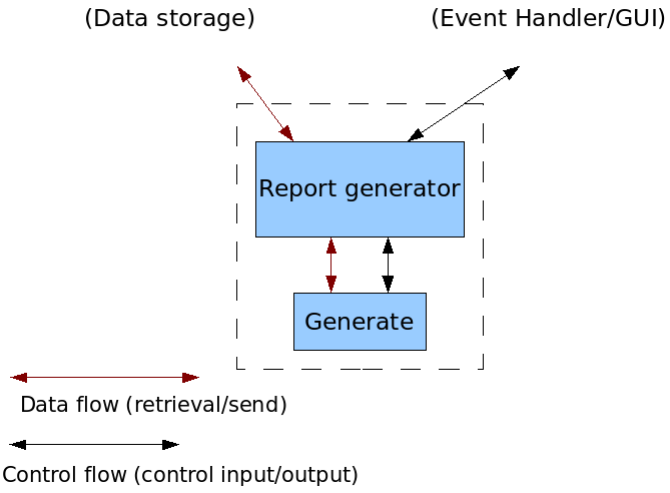


Figure 10: Control and data-flow model of the report generator component

3. Design considerations

3.1. Assumptions and dependencies

Operating systems

Our system is intended to be cross-platform, being at least able to run on the following operating systems:

- Windows XP
- GNU/Linux (Ubuntu 7.x)
- FreeBSD 6.2

Related software or hardware

Our system is not dependent on any specific hardware and will run on any hardware that will be able to run the operating systems listed above. The software itself is dependent on a few open-source libraries:

- *wxWidgets* (`http://www.wxwidgets.org/`)
- *TinyXML* (`http://www.grinninglizard.com/tinyxml/`)
- *VMime* (`http://www.vmime.org`)
- *Haru Free PDF Library* (`http://libharu.sourceforge.net/`)

End-user characteristics

The user of the system is not required to have great experience with computer software, the only thing the user will do is navigate through dialogs and press buttons. The user is assumed to be able to read standard English.

Possible and/or probable changes in functionality

The functionality and use of the external system RES used in our system could experience some changes during the implementation phase due to some research difficulties and lack of deep knowledge and good documentation about it.

3.2. General constraints

Character encodings

The main data input to the system are e-mails sent to the user (user requirements 2.1.1 and 2.1.2 section 4.a in [2]). E-mails can be encoded using a variety of different character encodings. To handle this all incoming data will be converted into *UTF-8* and all components of the systems should be capable to process UTF-encoded data.

Multi-platform

The system should build and run on a variety of operating systems (user requirement 7.1.2, section 4.a in [2]). *GCC* (`http://gcc.gnu.org/`) is the compiler that will be used for this project as it is ported to all supported operating systems. The *Windows* port of the standard library lacks some functionality so only a safe subset of the standard library can be used. At the moment wide character functions and localization functions are known to not work on *Windows*. All libraries used in the project must be implemented on all supported operating systems.

External system communication

The system will read and update information in the *RES* system (user requirements 6.1.1, 6.1.2 and 6.2.1 section 4.a in [2]). The *RES* system is not built to support this so the integration with it will be very weak. The communication will be implemented using small helper applications and intermediate data files.

Independent components

The system is divided into a set of different components. Each component will be developed by a single member of the group. The components will be developed independent of each other as far as possible. The public interface of each component must be very well defined and only define what is needed in the complete system. There is no easy method to do unit testing of the components based on their public interface so only tests on the completed application will be defined in this document.

Standards

The system must comply to a number of standards defined by network protocols and file formats. There are too many standards so the project will rely on third party libraries that implement all needed protocols and file formats.

4. Graphical User Interface

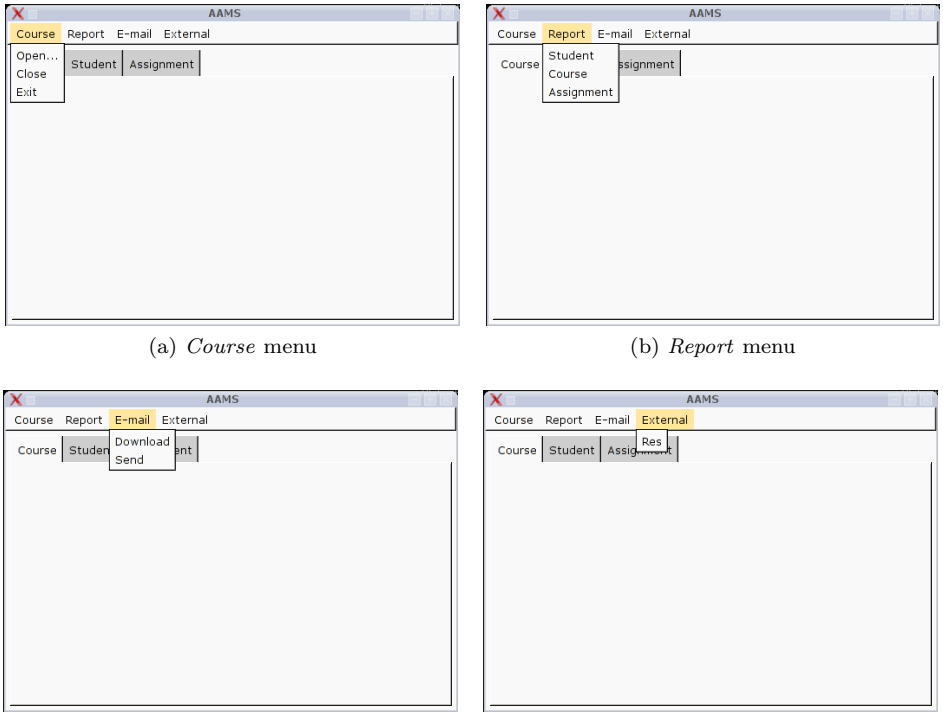
4.1. Overview of the User Interface

The graphical user interface in the program is organized in a single window with three different tabs and a menu in the top of the window.

The three tabs are *Course*, *Student* and *Assignment* (figure 12). In the *Course* tab (figure 12a) the general information about the course is presented such as the course name, course code and e-mail of the teacher. These general information fields for the course is editable. In the *Student* tab (figure 12b) and the *Assignment* tab (figure 12c) the user can show and edit the information about students and assignments. The *Student* tab contains a list of students for the course and the user is presented with two buttons, one for editing the properties of a student and one for viewing the assignments for a student (figure 13a). The *Assignment* tab contains a list of all the assignments in the course. In this tab there are also two buttons, one for editing the assignment called *Edit properties* and one for viewing the assignment for each student called *View assignment* (figure 13b). In the edit dialogs that will pop up when the user presses *Edit Properties* in the *Student/Assignment* tab (figure 14), this will have the option to edit the grade and add comments for students/assignments.

In the top of the window there is a menu with four items: *Course* (figure 11a), *Report* (figure 11b), *E-mail* (figure 11c) and *External* (figure 11d). In the *Course* menu the user can open and close different courses handled by the system. Opening another course will present the data for that course in the three tabs in the main window. In this menu there is also an *Exit* item that will terminate the program. The second menu is the *Report* menu which will create a report of either the students, the courses or the assignments. The next menu is the *E-mail* menu which contains the *Download* and *Send* items. The *Download* item will download all the e-mails sent to the system to the specified e-mail address. The *Send* item will send all the comments to the users by e-mail. The last menu is the *External* menu which will handle the communication with the external systems such as *RES*.

4.2. Screen shots



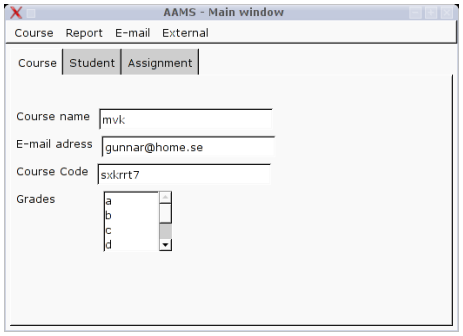
(a) *Course* menu

(b) *Report* menu

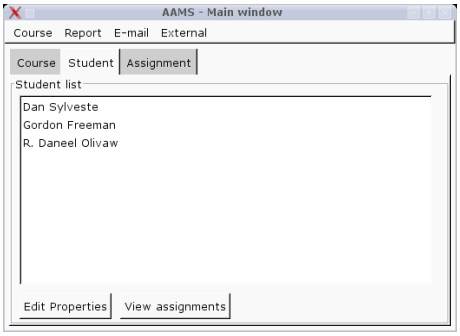
(c) *E-mail* menu

(d) *External* menu

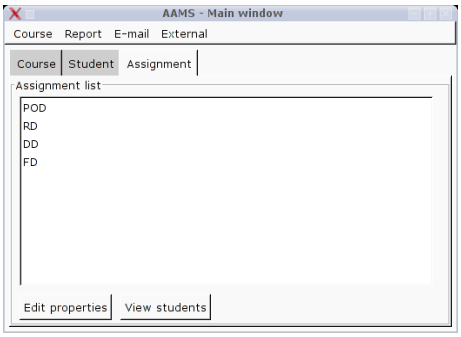
Figure 11: Screen shots showing the different menus



(a) *Course* tab

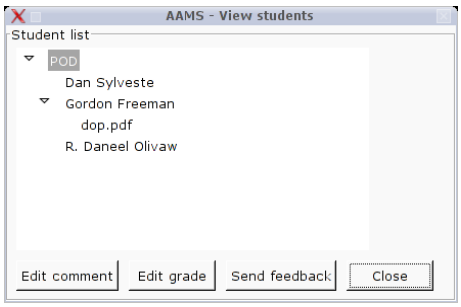


(b) *Student* tab

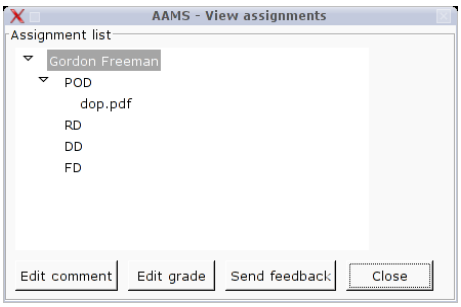


(c) *Assignment* tab

Figure 12: Screen shots showing the three main tabs

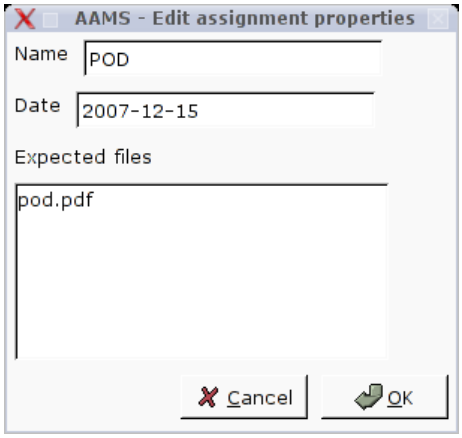


(a) *View students* form

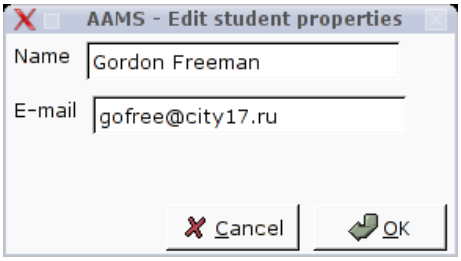


(b) *View assignments* form

Figure 13: Screen shots showing the *View students* and *View assignments* forms

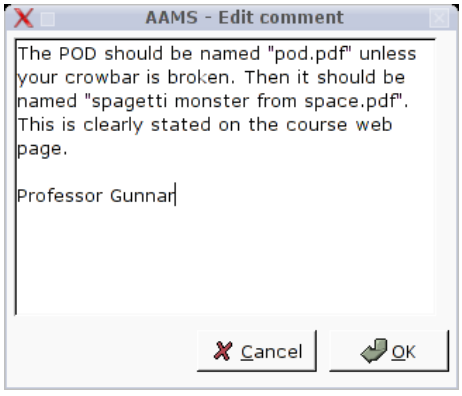


(a) *Edit assignment properties* form

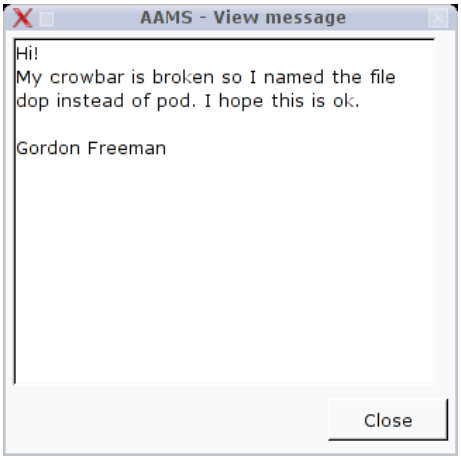


(b) *Edit student properties* form

Figure 14: Screen shots showing the properties edition forms for assignments and students



(a) *Edit comment* form



(b) *View message* form

Figure 15: Screen shots showing the edition of comments and view of messages forms

4.3. User Interface description

Main window

This form will be displayed at the beginning of the execution of the application, from the *OnInit* method of the *wxWidgets* Application Class.

- **Menus:**
  - ***Course***
    - ***Open...***
      A *click* event over this menu item will display a classical "open file" dialog box which will allow the user to select the course file to be opened.
    - ***Close***
      A *click* event over this menu item will close the currently opened course file.
    - ***Exit***
      A *click* event over this menu item will exit the application.
  - ***Report***
    - ***Student***
      A *click* event over this menu item will generate a report from the list of students of the currently opened course.
    - ***Course***
      A *click* event over this menu item will generate an overview report from the currently opened course.
    - ***Assignment***
      A *click* event over this menu item will generate a report from the list of assignments of the currently opened course.
  - ***E-mail***
    - ***Download***
      A *click* event over this menu item will start the process of connecting the specified e-mail server and download new received assignments from there.
    - ***Send***
      A *click* event over this menu item will start the process of connecting the specified e-mail server and sending all the generated information (comments) concerning the course assignments.
  - ***External***
    - ***RES***
      A *click* event over this menu item will run the RES plug-in communication process.
- **Tabs:**
  - ***Course* tab**
    - ***Course name* text box**
      A *text change* event from this control will automatically update the course file.
    - ***E-mail address* text box**
      A *text change* event from this control will automatically update the course file.
    - ***Course code* text box**
      A *text change* event from this control will automatically update the course file.

- ***Student* tab**
  - ***Student list* list box**
    A *double click* event over an item from this control will display the *View assignments* form for the selected item.
  - ***Edit properties* command button**
    A *click* event on this control will display the *Edit student properties* form for the item selected in the *Student list* list box.
  - ***View assignments* command button**
    A *click* event on this control will display the *View assignments* form for the item selected in the *Student list* list box.
- ***Assignment* tab**
  - ***Assignment list* list box**
    A *double click* event over an item from this control will display the *View students* form for the selected item.
  - ***Edit properties* command button**
    A *click* event on this control will display the *Edit assignment properties* form for the item selected in the *Assignment list* list box.
  - ***View students* command button**
    A *click* event on this control will display the *View students* form for the item selected in the *Assignment list* list box.

***View students* form**

This form will be displayed after a *double click* event on a student from the list of students of the *Student* tab which belongs to the main window.

- ***Student list* tree list box**
  A *click* event on an assignment item (first level of the tree list) will fold down a second-level list of students who have submitted that assignment. A *click* event on an student item (second level of the tree list) will fold down a third-level list of files that belong to the specified assignment and student. A *double click* event on a file item will launch the proper external application in order to open it.
- ***Edit comment* command button**
  A *click* event on this control will display the *Edit comment* form for the file item (third level) selected in the *Student list* tree list box.
- ***Edit grade* command button**
  A *click* event on this control will display a dialog box that allows the user to introduce a grade for the student item (second level) selected in the *Student list* tree list box.
- ***Send feedback* command button**
  A *click* event on this control will start the process of connecting the e-mail server and sending the edited comments for the assignments.
- ***Close* command button**
  A *click* event on this control will just close the *View students* form and will give back the focus to the main window.

***View assignments* form**

This form will be displayed after a *double click* event on an assignment from the list of assignments of the *Assignment* tab which belongs to the main window.

- ***Assignment list* tree list box**
  A *click* event on a student item (first level of the tree list) will fold down a second-level list of assignments that the selected student has submitted. A *click* event on an assignment item (second level of the tree list) will fold down a third-level list of files that belong to the specified assignment and student. A *double click* event on a file item will launch the proper external application in order to open it.
- ***Edit comment* command button**
  A *click* event on this control will display the *Edit comment* form for the file item (third level) selected in the *Assignment list* tree list box.
- ***Edit grade* command button**
  A *click* event on this control will display a dialog box that allows the user to introduce a grade for the assignment item (second level) selected in the *Assignment list* tree list box.
- ***Send feedback* command button**
  A *click* event on this control will start the process of connecting the e-mail server and sending the edited comments for the assignments.
- ***Close* command button**
  A *click* event on this control will just close the *View assignments* form and will give back the focus to the main window.

***Edit assignment properties* form**

This form will be displayed after a *click* event on the *Edit properties* command button from the *Assignment* tab which belongs to the main window.

- ***Name* text box**
  A *text change* event from this control will automatically update the course file.
- ***Date* text box**
  A *text change* event from this control will automatically update the course file.
- ***Expected files* list box** A *double click* event from this control will add a new item in the list.
- ***OK* command button**
  A *click* event on this control will close the *Edit assignment properties* form and will give back the focus to the main window, saving the changes.
- ***Cancel* command button**
  A *click* event on this control will close the *Edit assignment properties* form and will give back the focus to the main window, discarding the changes.

***Edit student properties* form**

This form will be displayed after a *click* event on the *Edit properties* command button from the *Student* tab which belongs to the main window.

- ***Name* text box**
  A *text change* event from this control will automatically update the course file.
- ***E-mail* text box**
  A *text change* event from this control will automatically update the course file.
- ***OK* command button**
  A *click* event on this control will close the *Edit student properties* form and will give back the focus to the main window, saving the changes.
- ***Cancel* command button**
  A *click* event on this control will close the *Edit student properties* form and will give back the focus to the main window, discarding the changes.

***Edit comment* form**

This form will be displayed after a *click* event on the *Edit comment* command button from the *View students* and *View assignments* forms.

- ***Comment* text box**
  A *text change* event from this control will automatically update the course file.
- ***OK* command button**
  A *click* event on this control will close the *Edit comment* form and will give back the focus to the main window, saving the changes.
- ***Cancel* command button**
  A *click* event on this control will close the *Edit comment* form and will give back the focus to the main window, discarding the changes.

***View message* form**

This form will be displayed after a *click* event on the *View message* command button from the *View students* and *View assignments* forms.

- ***Message* text box**
  This control is not able to generate any event.
- ***Close* command button**
  A *click* event on this control will close the *View message* form and will give back the focus to the main window.

4.4. Major reports description

We will list here the UI error reports, referencing to the different Use Cases from the Requirements Document ([2]). All these messages will be shown to the user in a standard dialog box:

- **1.1 Extensions**
  - 3a Message: "Invalid course name"
- **1.2 Extensions**
  - 3a1 Message: "Invalid location"
  - 4a1 Message: "Invalid format in file"
- **1.3 Extensions**
  - 4a1 Message: "Invalid assignment name"
- **2.1 Extensions**
  - 2a Message: "No files found!""
- **2.2 Extensions**
  - 3a Message: "Invalid login/pass"
  - Message: "Connection could not be established"
  - Message: "Connection timed out"
- **3.1 Extensions**
  - 3a1 Message: "Invalid file format"
- **3.3 Extensions**
  - 5a1 Message: "Invalid grade"
- **4.2 Extensions**
  - 2a1 Message: "Invalid login/pass
  - Message: "Connection could not be established"
  - Message: "Connection timed out"
- **5.1 Extensions**
  - 3a1 Message: "Course was not found!"
  - Message: "Invalid course name"
- **5.2 Extensions**
  - 3a1 Message: "Student was not found!"
  - Message: "Invalid student name"
- **5.3 Extensions**
  - 3a1 Message: "Assignement was not found"

- Message: "Invalid assignment name"

- **6.1 Extensions**
  - 1a1 Message: "Invalid course name"

- **6.2 Extensions**
  - 2a1 Message: "No grades assigned"
  - 4a1 Message: "Invalid course"

5. Design details

5.1. Class Responsibility Collaborator (CRC) Cards

Student communication component

Class mailbox

Responsibilities	Collaborators
Download and save new assignments	pop3Transport
Compose and send feedback	smtpTransport
Compose and send confirmation e-mails	mail
Send previously stored messages	validator
Store messages that could not be sent	CourseDataStorage
Validate e-mails	UnsentEmailStorage
	GlobalStorage

Class pop3Transport

Responsibilities	Collaborators
Download message headers from POP3 server	mail
Download messages from POP3 server	mail

Class smtpTransport

Responsibilities	Collaborators
Send messages to SMTP server	mail

Class mail

Responsibilities	Collaborators
Store e-mail data for use by other components	attachment
Associate attachments with a message	

Class attachment

Responsibilities	Collaborators
Store e-mail attachments for use by other components	

Class validator

Responsibilities	Collaborators
validate e-mail headers	

External communication and RES communication components

Class externalCom

Responsibilities	Collaborators
Create grades exchange files from AAMS courses	CourseDataStorage
Import lists of students from exchange files to AAMS	External system communication modules

Class RESCom

Responsibilities	Collaborators
Create students exchange files from RES system courses	RES system
Import grades from exchange files to RES system	externalCom

GUI/Event handler component

Class AAMSGUI

Responsibilities	Collaborators
Manage the AAMS GUI	wxWidgets

Class GUIcreator

Responsibilities	Collaborators
Define and show GUI	wxWidgets
Define event table	GUI controller

Class GUIcontroller

Responsibilities	Collaborators
Manage GUI events	wxWidgets
	ErrorManager
	WindowManager
	FileManager

Class ErrorManager

Responsibilities	Collaborators
Display AAMS error messages	WindowManager

Class WindowManager

Responsibilities	Collaborators
Coordinate the display of different windows	wxWidgets

Class FileManager

Responsibilities	Collaborators
Show files by calling the proper external application	wxWidgets

Data storage component

Class CourseDataStorage

Responsibilities	Collaborators
Load the data for a specified course	CourseData
Save data for the course	EmailAccountPOP3Data
Add new data for a course	EmailAccountSMTPData
Edit data for a course	XMLCourseDataRead
Remove data for a course	XMLCourseDataWrite

Class CourseData

Responsibilities	Collaborators
Hold data for a course	StudentData
	FileInfoData
	AssignmentInfoData

Class StudentData

Responsibilities	Collaborators
Hold data for a student in the course	AssignmentData

Class AssignmentData

Responsibilities	Collaborators
Hold the assignment data for a student	FileData

Class FileData

Responsibilities	Collaborators
Hold the file data for an assignment	

Class FileInfoData

Responsibilities	Collaborators
Hold information about files in assignments	

Class AssignmentInfoData

Responsibilities	Collaborators
Hold information about assignments in a course	

Class EmailAccountPOP3Data

Responsibilities	Collaborators
Hold information about the POP3 server settings with login information	

Class EmailAccountSMTPData

Responsibilities	Collaborators
Hold information about the SMTP server settings with login information	

Class XMLCourseDataRead

Responsibilities	Collaborators
Read the XML file containing the course data	CryptUtil

Class XMLCourseDataWrite

Responsibilities	Collaborators
Write the course data to the XML file containing the course data	CryptUtil

Class UnsentEmailsStorage

Responsibilities	Collaborators
Load data for unsent emails from file	UnsentEmailsData
Save data for unsent emails from file	XMLUnsentEmailsDataRead
	XMLUnsentEmailsDataWrite

Class UnsentEmailsData

Responsibilities	Collaborators
Hold data for unsent emails	

Class XMLUnsentEmailsDataRead

Responsibilities	Collaborators
Read the XML file containing the unsent emails data	CryptUtil

Class XMLUnsentEmailsDataWrite

Responsibilities	Collaborators
Read the XML file containing the unsent emails data	CryptUtil

Class CryptUtil

Responsibilities	Collaborators
Encrypt	
Decrypt	

Report generator component

Class CreatePdf

Responsibilities	Collaborators
Generate a pdf file based on received data	CourseData PDFGenerateType

Class PDFGenerateType

Responsibilities	Collaborators
Defines which type of report to generate	AAMSGUI

5.2. Class diagrams

Student communication component

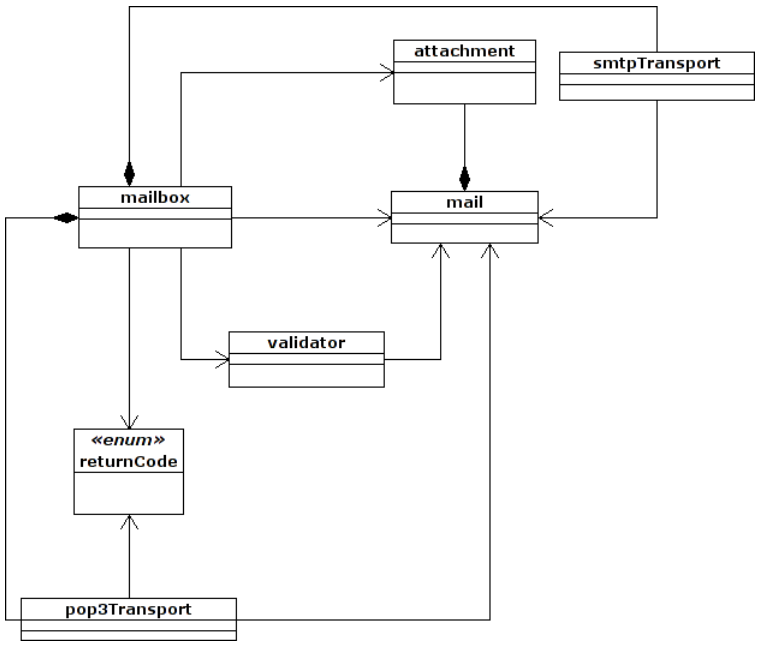


Figure 16: Class diagram for the student communication component

External communication and RES communication components

In this simple diagram (figure 17), the two ways slashed line represents the dependency between both classes/components. It is important to remark that the scope of the *RESCom* component is extern to the system, i.e. it will be implemented as an independent application. The communication between the two components will be possible through temporary exchange files and operative system execution calls (where the *RESCom* component will act always as a passive server and the AAMS system will act as a client, running the external *RESCom* applications when necessary).

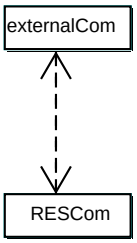


Figure 17: Class diagram for the external communication and RES communication components

GUI/Event handler component

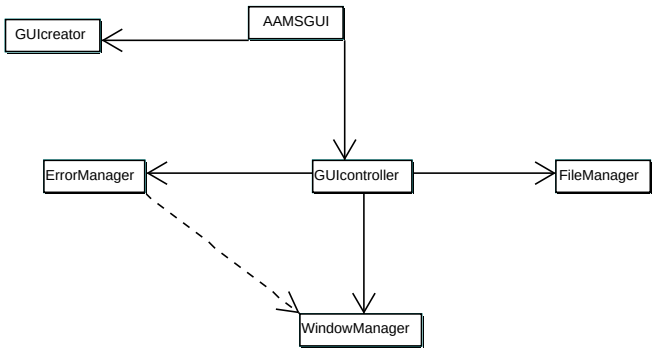


Figure 18: Class diagram for the GUI/Event handler component

Data storage component

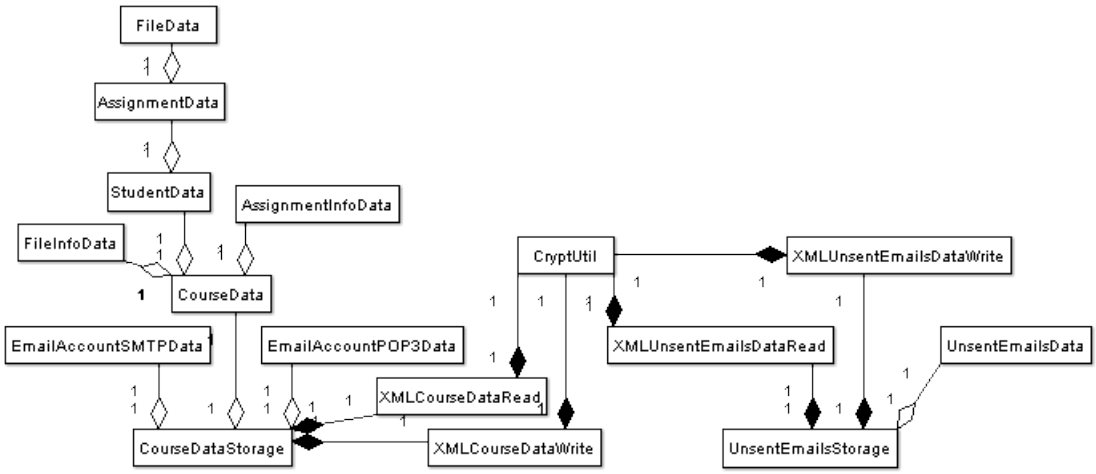


Figure 19: Class diagram for the data storage component

Report generator component

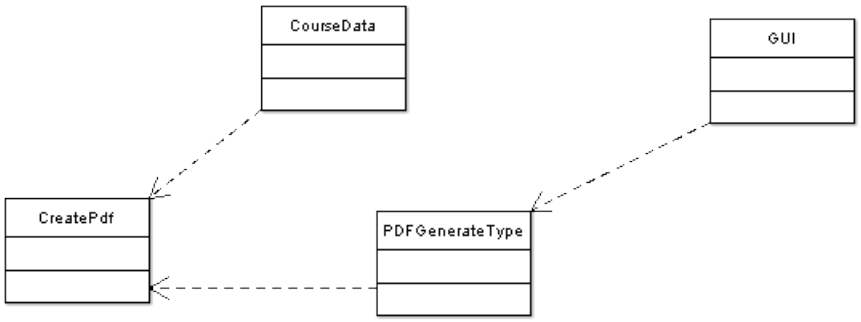


Figure 20: Class diagram for the report generator component

5.3. State charts

Due to the fact that the state model of almost every component in the system is trivial, only one state chart, relative to the GUI component, has been considered necessary in this subsection.

GUI/Event handler component

The next state chart (figure 21) shows the different GUI states (represented by forms or windows and tabs inside these) and the transitions between them triggered by the generation of user events:

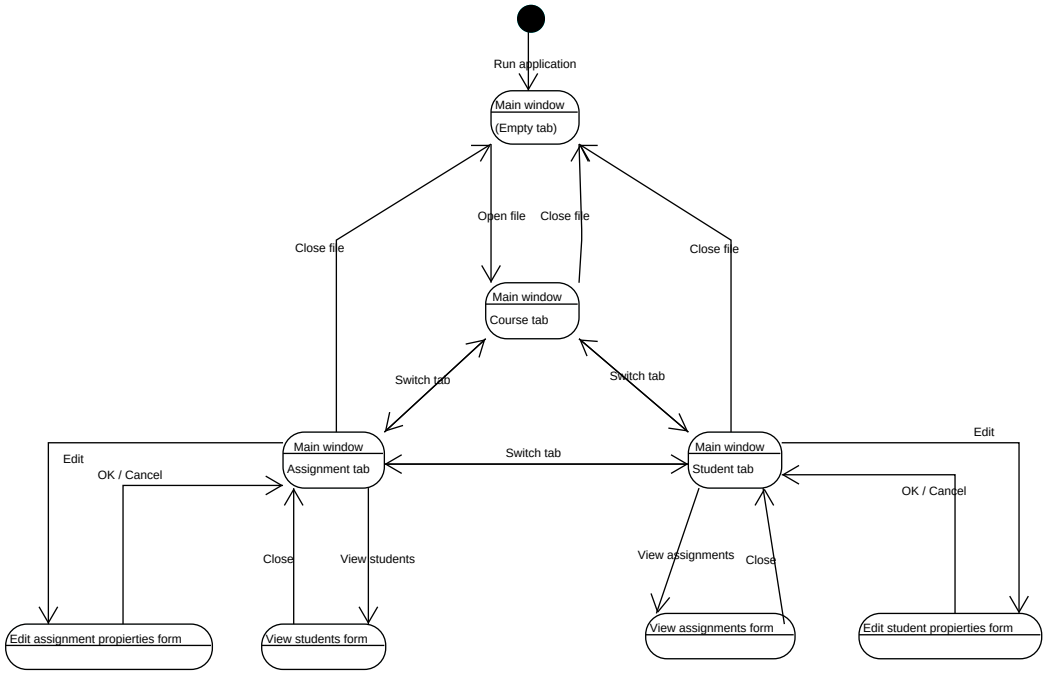


Figure 21: State chart for the GUI presentation

5.4. Interaction diagrams

Student communication component

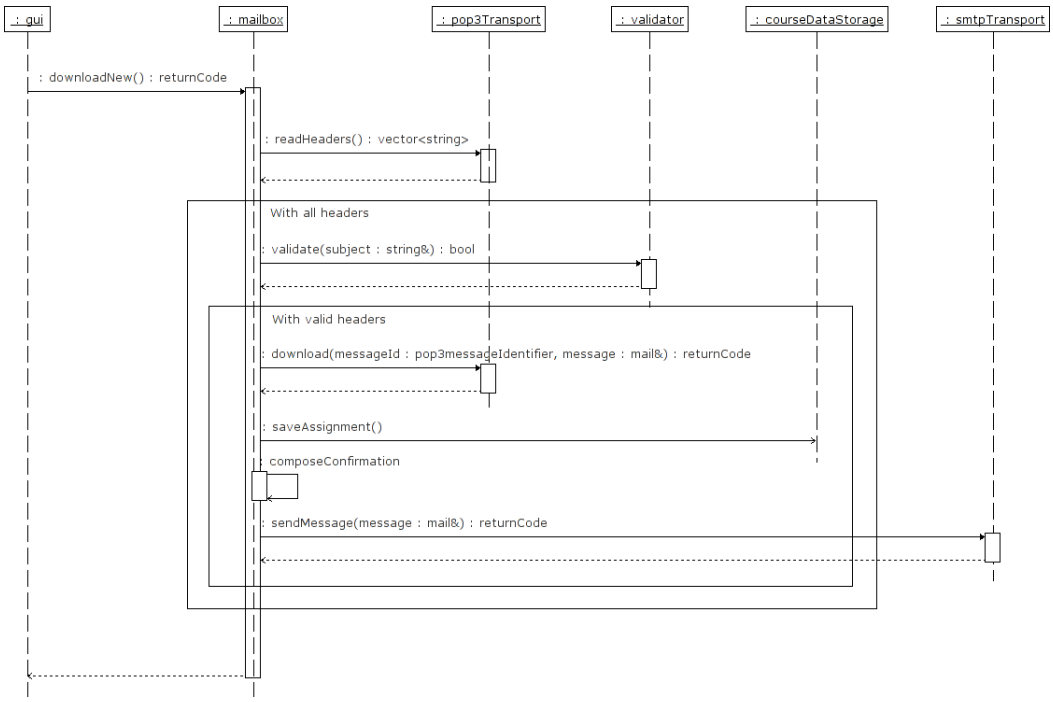


Figure 22: Interaction diagram for the download of an assignment

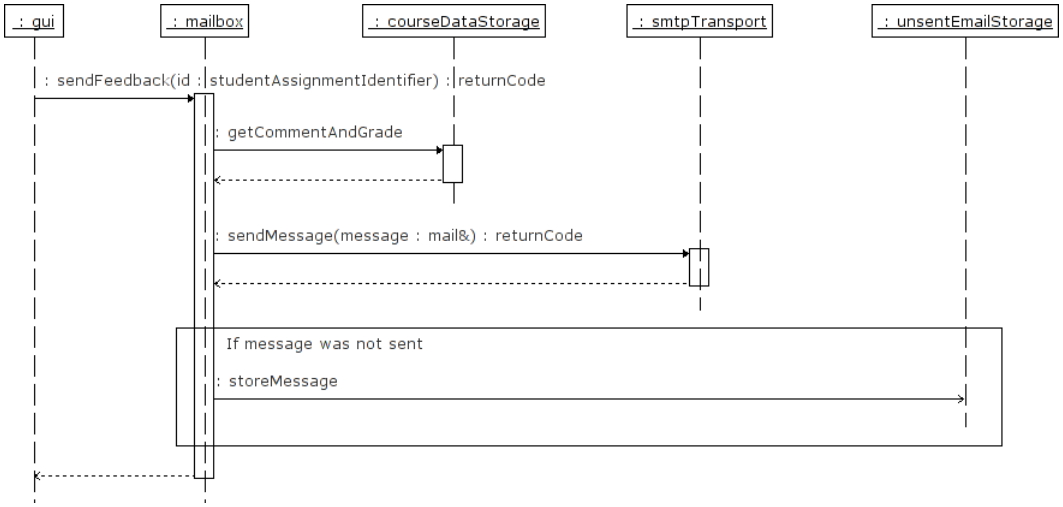


Figure 23: Interaction diagram for the sending of feedback

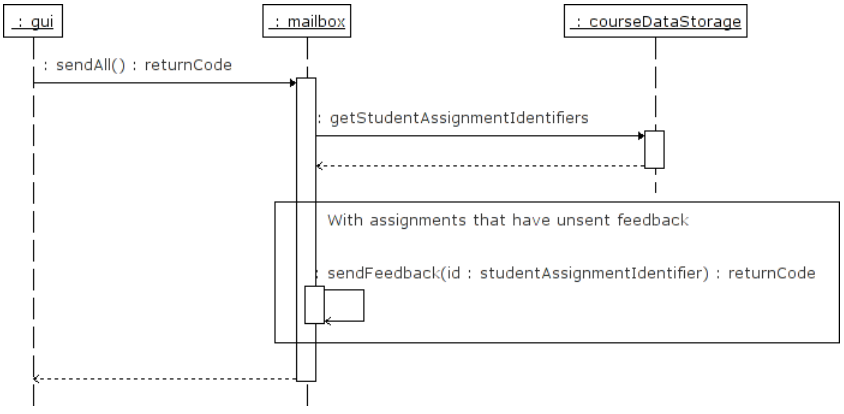


Figure 24: Interaction diagram for the sending of all unsent feedback

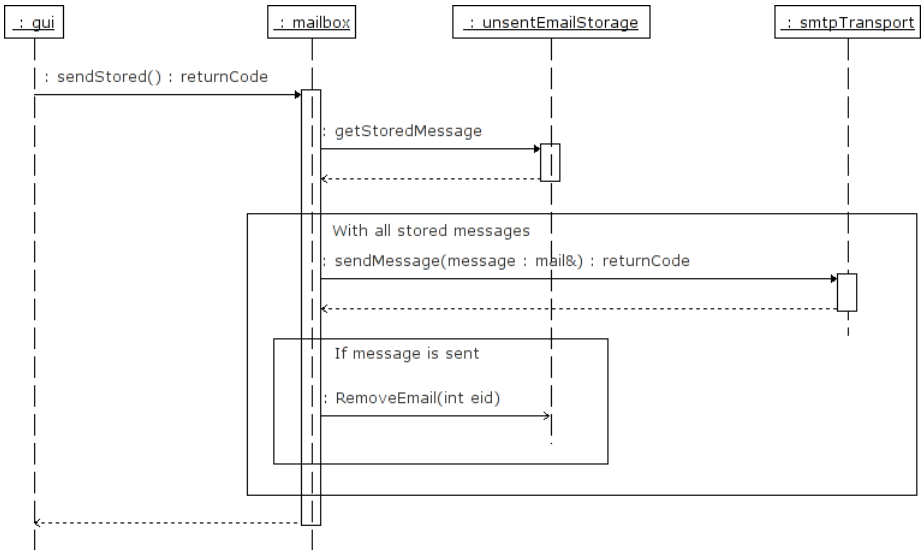


Figure 25: Interaction diagram for the sending of all stored messages

External communication and RES communication components

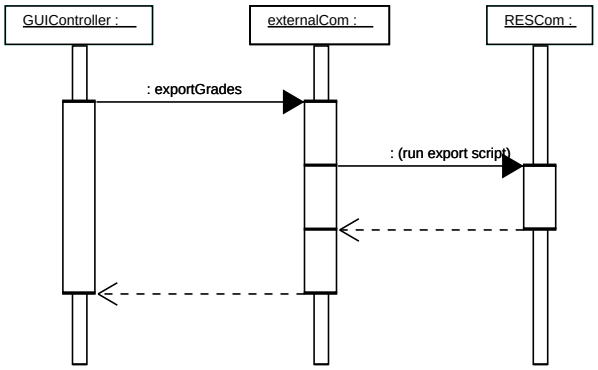


Figure 26: Interaction diagram for the exporting of grades

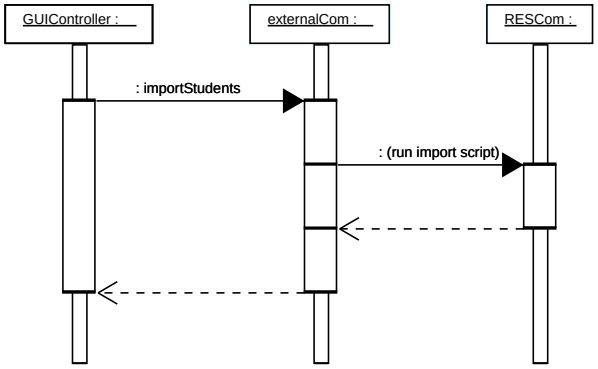


Figure 27: Interaction diagram for the importing of grades

GUI/Event handler component

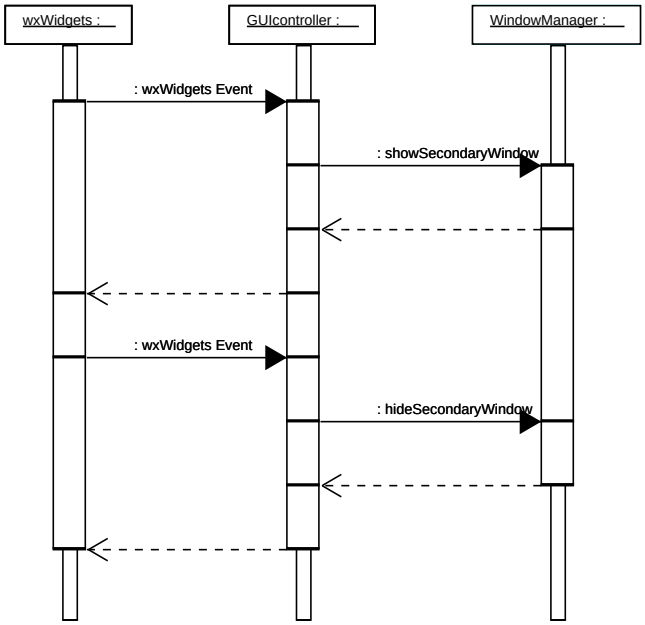


Figure 28: Interaction diagram for the opening and closing of a secondary window as a result of user events

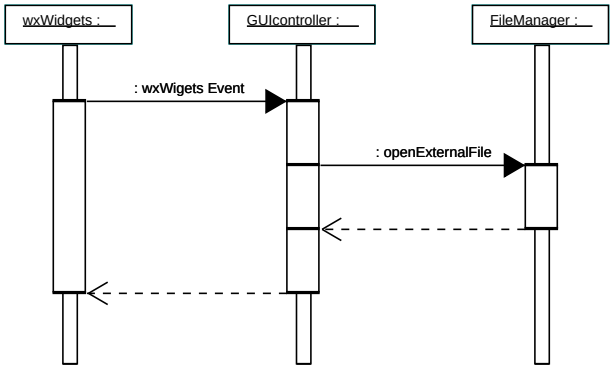


Figure 29: Interaction diagram for the opening of a file with an external application as a result of a user event

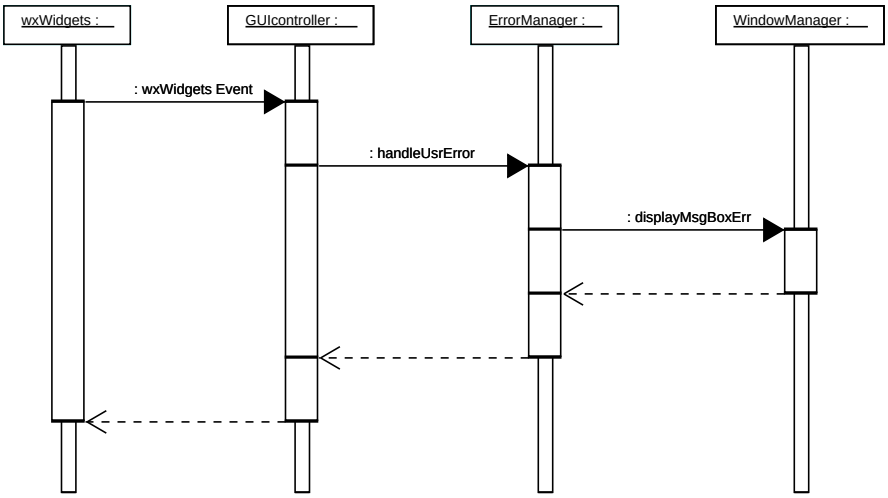


Figure 30: Interaction diagram for the handling of a user error

Data storage component

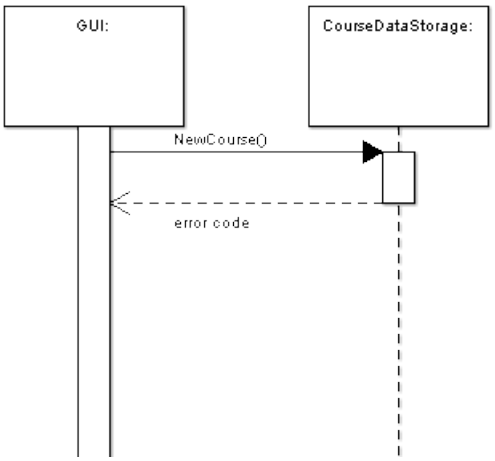


Figure 31: Interaction diagram for the creation of a new course

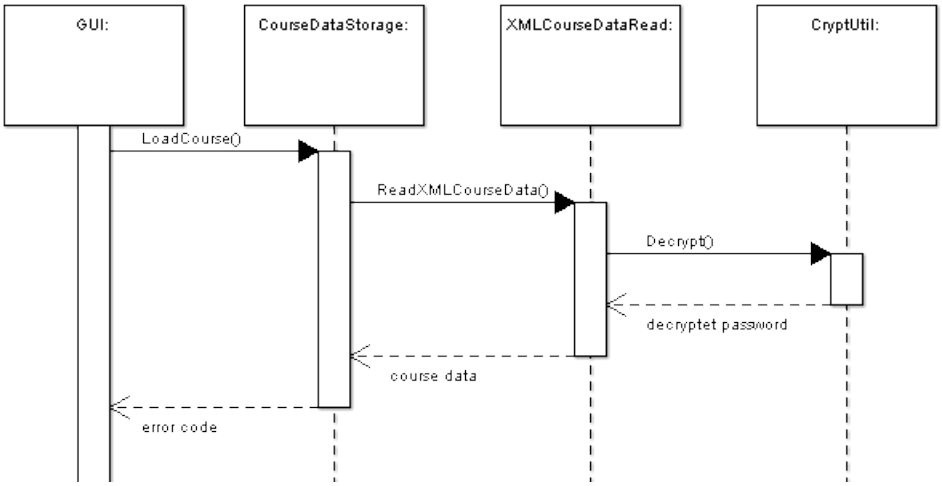


Figure 32: Interaction diagram for the opening of a course

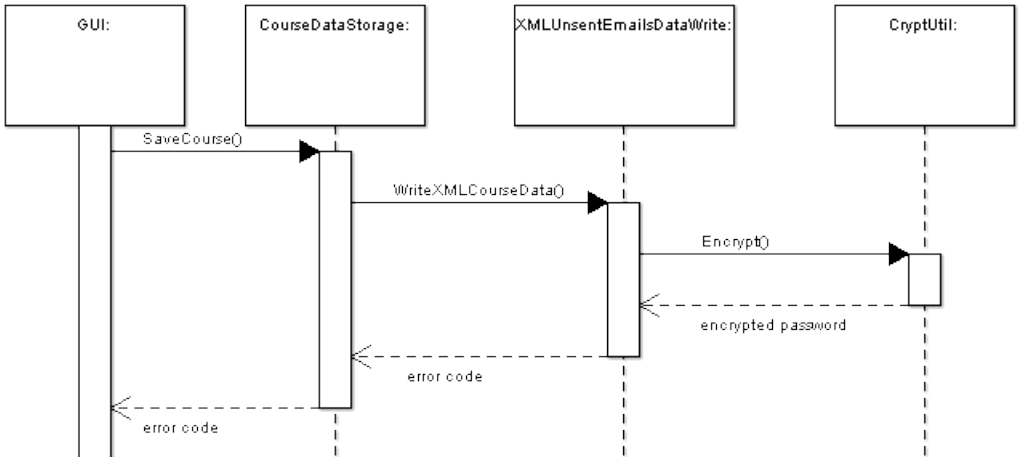


Figure 33: Interaction diagram for the saving of a course

Report generator component

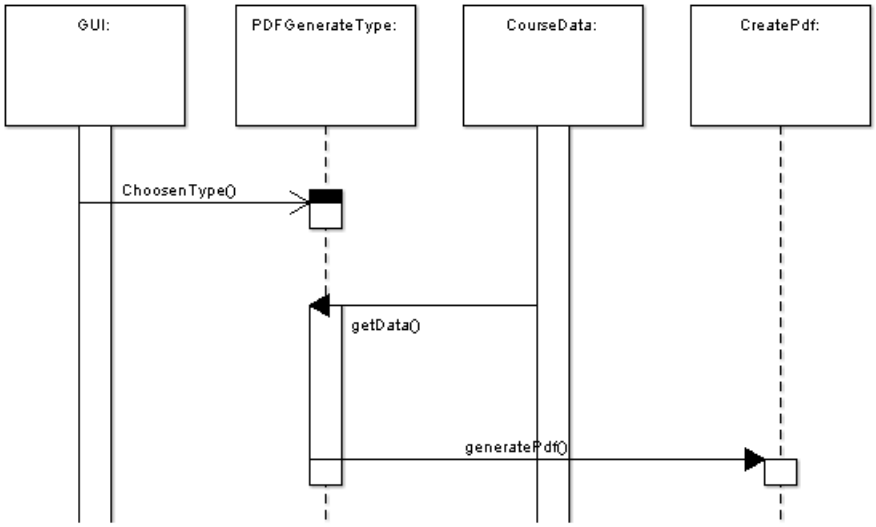


Figure 34: Interaction diagram for the generation of a report in PDF format

5.5. Detailed Design

Global data model

The organization of the data managed by the system is described in a logical level with the next Entity-Relationship (ER) diagrams (where the boxes indicate entities and the "forks" in the edges indicate multiplicity larger than 1) and attribute descriptions for every class:

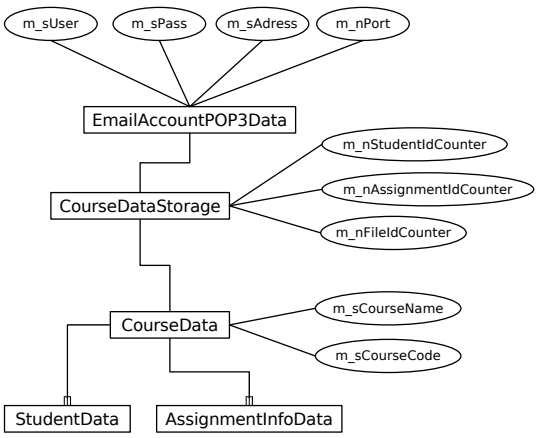


Figure 35: ER diagram for all the information that must be stored for a course

class CourseDataStorage

- *CourseData m_courseData*: all the information that belongs to a course (see below).
- *int m_nStudentIdCounter*: number of students in a course.
- *int m_nAssignmentIdCounter*: number of defined assignments for a course.
- *int m_nFileIdCounter*; number of files for a course.
- *EmailAccountPOP3Data m_POP3*: information about the selected POP3 account for a course (see below).

class EmailAccountPOP3Data

- *std::string m_sUser*: user name for the POP3 account.
- *std::string m_sPass*: password for the POP3 account.
- *std::string m_sAddress*: address of the POP3 server.
- *int m_nPort*: port for the POP3 protocol in the target server.

class CourseData

- *std::wstring m_sCourseName*: name of the course.
- *std::wstring m_sCourseCode*: code of the course.
- *std::list<StudentData> m_StudentData*: list of students and their assignments (see below).
- *std::list<AssignmentInfoData> m_assignmentInfoData*: list of defined assignments (see below).

class StudentData

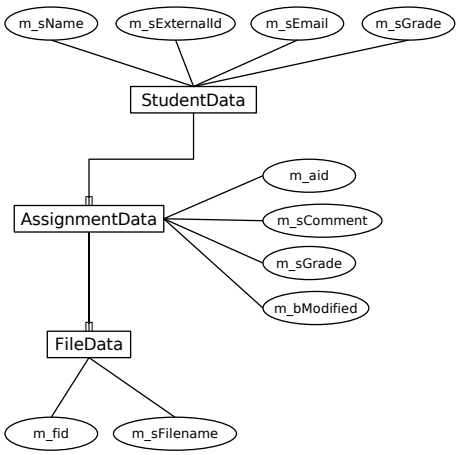


Figure 36: ER diagram showing the data model for students and assignments

- *int m_sid*: internal identifier of the student.
- *std::wstring m_sName*: name of the student.
- *std::wstring m_sExternalId*: external (institutional) identifier of the student.
- *std::wstring m_sEmail*: e-mail address of the student.
- *std::wstring m_sGrade*: global grade of the student.
- *std::list<AssignmentData> m_assignmentData*: list of assignments of the student (see below).

class AssignmentData

- *int m_aid*: internal identifier of the assignment.
- *std::wstring m_sComment*: teacher's comment on the assignment.
- *std::wstring m_sGrade*: grade of the assignment.
- *bool m_bModified*: specifies if the comments or the grade have been modified since the last time it was notified to the student.
- *std::list<FileData> m_fileData*: list of files that the assignment contains (see below).

class FileData

- *int m_fid*: internal identifier of the file.
- *std::wstring m_sFilename*: path of the file.

class AssignmentInfoData

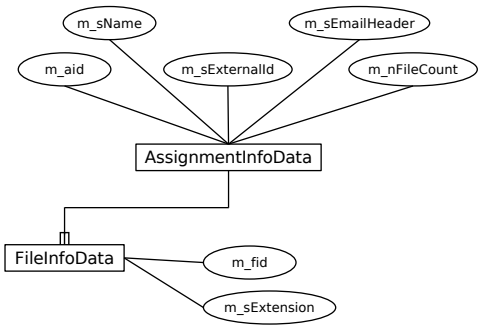


Figure 37: ER diagram showing the data model for the assignment definitions in a course

- *int m_aid*: internal identifier of the assignment definition.
- *std::wstring m_sName*: name of the assignment.
- *std::wstring m_sExternalId*: external (institutional) identifier of the assignment.
- *std::wstring m_sEmailHeader*: expected e-mail subject for sending the assignment.
- *int m_nFileCount*: expected number of files attached to the assignment.
- *std::list<FileInfoData> m_fileInfoData*: list of expected file formats of the assignment (see below).

class FileInfoData

- *int m_fid*: internal identifier of the file format.
- *std::wstring m_sExtension*: file extension.

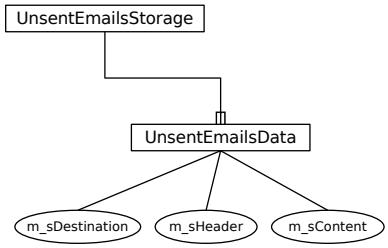


Figure 38: ER diagram showing the data model for the unsent e-mails

class UnsentEmailsStorage

- *int m_nEmailIdCounter*: number of unsent e-mails.

- *std::list<UnsentEmailsData> m_unsentEmailsData*: list of unsent e-mails (see below).

class UnsentEmailsData

- *string m_sDestination*: destination address of the unsent e-mail.
- *string m_sHeader*: header of the unsent e-mail.
- *string m_sContent*: content of the unsent e-mail.

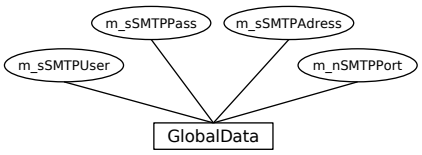


Figure 39: ER diagram showing the data model for the global settings of the system

class GlobalData

- *std::string m_sSMTPUser*: user name for the SMTP account.
- *std::string m_sSMTPPass*: password for the SMTP account.
- *std::string m_sSMTPAdress*: address of the SMTP server.
- *int m_nSMTPPort*: port for the SMTP protocol in the target server.

Student communication component

- **Method Name:** `mailbox::mailbox(GlobalStorage& globalData, UnsentEmailsStorage& unsentData, CourseDataStorage& courseData)`
- **Parameters:**
  - globalData. A reference to a globalStorage object.
  - unsentData. A reference to a UnsentEmailsStorage object.
  - courseData. A reference to a CourseDataStorage object.
- **Description:** Initializes a new mailbox object.
- **Data structure and tables it accesses:** This method does not access any data from the storage references.
- **Pre-conditions:** None
- **Post-conditions:** The mailbox object is initialized.
- **Called by:** Main application.
- **Calls:** None

- **Method Name:** `returnCode mailbox::downloadNew()`
- **Return Value:** One of the following:
  - NOERROR The method succeeded but no assignments was imported
  - MAIL_NEWASSIGNMENT The method succeeded and assignments was imported
  - MAIL_SMTPUNAVAIL The SMTP server was unavailable
  - MAIL_SMTPREFUSE The SMTP sever refused connection
  - MAIL_POPUNAVAIL The POP3 server was unavailable
  - MAIL_POPREFUSE The POP3 sever refused connection
  - MAIL_STORAGEERROR An error occurred while storing assignments, possible loss of data
- **Description:** Imports assignments from e-mail and sends confirmation e-mails. This method is the implementation of user requirement: 2.1.1, 2.1.2, 2.2.2, 4.1.1 and 4.1.2
- **Data structure and tables it accesses:**
  - SMTP account information from globalData.
  - POP3 account information from courseData
  - AssignmentInfoData from courseData
  - StudentData from courseData
  - May store data in unsentData
- **Pre-conditions:** Accurate data loaded into storage objects
- **Post-conditions:** New assignments are downloaded and stored into courseData. Confirmation e-mails are sent or stored in unsentData

- **Called by:** Main application.
- **Calls:** Functions in the storage objects.

- **Method Name:** `returnCode mailbox::sendStored()`
- **Return Value:** One of the following:
  - NOERROR The method succeeded
  - MAIL_SMTPUNAVAIL The SMTP server was unavailable
  - MAIL_SMTPREFUSE The SMTP sever refused connection
  - MAIL_STORAGEERROR An error occurred while accessing unsentData
- **Description:** Sends stored e-mails. This method is required for implementation of user requirements: 4.1.1, 4.1.2 and 4.2.1.
- **Data structure and tables it accesses:**
  - SMTP account information from globalData.
  - E-mail data in unsentData
- **Pre-conditions:** Accurate data loaded into storage objects
- **Post-conditions:** Unsent e-mails are sent or stored in unsentData
- **Called by:** Main application.
- **Calls:** Functions in the storage objects.

- **Method Name:** `returnCode mailbox::sendAll()`
- **Return Value:** One of the following:
  - NOERROR The method succeeded
  - MAIL_SMTPUNAVAIL The SMTP server was unavailable
  - MAIL_SMTPREFUSE The SMTP sever refused connection
  - MAIL_STORAGEERROR An error occurred while accessing courseData or unsentData
- **Description:** Sends feedback, composed of a comment and a grade, to all students on modified assignments. This method is part of the implementation of user requirement. 4.2.1.
- **Data structure and tables it accesses:**
  - SMTP account information from globalData
  - StudentData from courseData
  - May store data in unsentData
- **Pre-conditions:** Accurate data loaded into storage objects.
- **Post-conditions:** Feedback e-mails are composed and sent or stored in unsentData
- **Called by:** Main application.
- **Calls:** mailbox::sendFeedback(int sID, int aID) Functions in the storage objects.

- **Method Name:** `returnCode mailbox::sendFeedback(int sID, int aID)`
- **Parameters:**
  - sID. Integer to identify a student in courseData.
  - aID Integer to identify the students assignment.
- **Return Value:** One of the following:
  - NOERROR The method succeeded
  - MAIL_SMTPUNAVAIL The SMTP server was unavailable
  - MAIL_SMTPREFUSE The SMTP sever refused connection
  - MAIL_STORAGEERROR An error occurred while accessing courseData or unsentData
- **Description:** Sends feedback, composed of a comment and a grade, to a student on a specified assignment. This method is the implementation of user requirements: 4.2.1 and 4.2.2
- **Data structure and tables it accesses:**
  - SMTP account information from globalData.
  - StudentData from courseData
  - May store data in unsentData
- **Pre-conditions:** Accurate data loaded into storage objects.
- **Post-conditions:** A feedback e-mail is composed and sent or stored in unsentData
- **Called by:** Main application and mailbox
- **Calls:** Functions in the storage objects.

- **Method Name:** `void mail::setAdress(wstring address)`
- **Parameters:** address. A string containing an e-mail address.
- **Description:** Sets the address in a mail object.
- **Pre-conditions:** None
- **Post-conditions:** The mail objects address is changed.

- **Method Name:** `void mail::setSubject(wstring subject)`
- **Parameters:** address. A string containing an e-mail subject.
- **Description:** Sets the subject in a mail object.
- **Pre-conditions:** None
- **Post-conditions:** The mail objects subject is changed.

- **Method Name:** `void mail::setMessage(wstring message)`

- **Parameters:** address. A string containing an e-mail message.
- **Description:** Sets the message in a mail object.
- **Pre-conditions:** None
- **Post-conditions:** The mail objects message is changed.

- **Method Name:** `void mail::addAttachment(wstring filename, char* data, int size)`
- **Parameters:**
  - filename. The name of the attached file
  - data. The data in the file.
  - size The size of data in bytes
- **Description:** Adds a attachment to a mail object.
- **Pre-conditions:** data points to allocated memory of the size of the parameter size
- **Post-conditions:** An attachment object is added to the mail objects attachments.

- **Method Name:** `wstring mail::getAdress()`
- **Return Value:** A string containing the address of a mail object. If the mail objects address is uninitialized the empty string is returned.
- **Description:** Retrieves the address of a mail object.
- **Pre-conditions:** None
- **Post-conditions:** None

- **Method Name:** `wstring mail::getSubject()`
- **Return Value:** A string containing the subject of a mail object. If the mail objects subject is uninitialized the empty string is returned.
- **Description:** Retrieves the subject of a mail object.
- **Pre-conditions:** None
- **Post-conditions:** None

- **Method Name:** `wstring mail::getMessage()`
- **Return Value:** A string containing the message of a mail object. If the mail objects message is uninitialized the empty string is returned.
- **Description:** Retrieves the message of a mail object.
- **Pre-conditions:** None

- **Post-conditions:** None

- **Method Name:** `int mail::countAttachment()`
- **Return Value:** A integer with the number of files attached to a mail object.
- **Description:** Retrieves the number of files attached to a mail object.
- **Pre-conditions:** None
- **Post-conditions:** None

- **Method Name:** `attachment* mail::getAttachment(int idx)`
- **Parameters:** idx. The desired attachment index in the mail object.
- **Return Value:** A pointer to the attachment pointed to by the parameter idx. If there are no attachment with the given index zero is returned.
- **Description:** Retrieves attachments from a mail object.
- **Pre-conditions:** None
- **Post-conditions:** None

- **Method Name:** `void mail::clear()`
- **Description:** Clears address, subject and message in a mail object. The attachments are deleted.
- **Pre-conditions:** None
- **Post-conditions:** address, subject and message are set to the empty string. Attachments are deleted.

- **Method Name:** `attachment::attachment(wstring filename, int size, char* data)`
- **Parameters:**
  - filename. The name of the attachment.
  - size. The size of the attachment in bytes.
  - data The attachment data.
- **Description:** Initializes an attachment object. The memory pointed to by data will be deallocated when the attachment object is destroyed.
- **Pre-conditions:** data points to allocated memory of the size of the parameter size
- **Post-conditions:** The attachment object is initialized
- **Calls:** None

- **Method Name:** `wstring attachment::getFilename()`
- **Return Value:** A string containing the filename of an attachment.
- **Description:** Retrieves the filename of an attachment.
- **Pre-conditions:** None
- **Post-conditions:** None
- **Calls:** None

- **Method Name:** `int attachment::getSize()`
- **Return Value:** The size of the attachment data in bytes.
- **Description:** Retrieves the size of an attachment.
- **Pre-conditions:** None
- **Post-conditions:** None
- **Calls:** None

- **Method Name:** `char* attachment::getData()`
- **Return Value:** A pointer to attachment data.
- **Description:** Retrieves the data from an attachment.
- **Pre-conditions:** None
- **Post-conditions:** None
- **Calls:** None

- **Method Name:** `validator::validator(CourseDataStorage& courseData)`
- **Parameters:** courseData. A reference to a CourseDataStorage object, used to validate e-mails.
- **Description:** Initializes a validator object.
- **Data structure and tables it accesses:** This method does not access any data in the courseData object.
- **Pre-conditions:** None.
- **Post-conditions:** The validator object is initialized.
- **Called by:** mailbox
- **Calls:** None

- **Method Name:** `bool validator::validate(wstring subject)`

- **Parameters:** subject. A string containing a e-mail subject line, used to validate the e-mail.
- **Return Value:** true if the e-mail can be related to a student and an assignment in courseData. Otherwise false.
- **Description:** Validates e-mail headers. This method is the implementation of user requirement: 2.1.2, 2.1.3.
- **Data structure and tables it accesses:** courseData
- **Pre-conditions:** Accurate data loaded into storage objects.
- **Post-conditions:** None
- **Called by:** mailbox
- **Calls:** Functions in courseData.

- **Method Name:** `smtpTransport::smtpTransport(GlobalStorage& globalData)`
- **Parameters:** globalData. A reference to a GlobalStorage object, used to get account information.
- **Description:** Initializes a smtpTransport object.
- **Data structure and tables it accesses:** SMTP account information in globalData.
- **Pre-conditions:** Accurate data loaded into storage objects.
- **Post-conditions:** The smtpTransport object is initialized.
- **Called by:** mailbox
- **Calls:** None

- **Method Name:** `returnCode smtpTransport::sendMessage(mail& message)`
- **Parameters:** message. A reference to a mail object.
- **Return Value:** One of the following:
  - NOERROR The method succeeded
  - MAIL_SMTPUNAVAIL The SMTP server was unavailable
  - MAIL_SMTPREFUSE The SMTP sever refused connection
- **Description:** Sends a e-mail message stored in a mail object.
- **Pre-conditions:** The mail object holds a e-mail.
- **Post-conditions:** If no error occurred the e-mail is sent.
- **Called by:** mailbox
- **Calls:** Functions in the e-mail library

- **Method Name:** `pop3Transport::pop3Transport(EmailAccountPOP3Data& pop3Data)`

- **Parameters:** pop3Data. A reference to a EmailAccountPOP3Data object, used to get account information.
- **Description:** Initializes a pop3Transport object.
- **Data structure and tables it accesses:** pop3 account information in pop3Data.
- **Pre-conditions:** Accurate data loaded into pop3Data.
- **Post-conditions:** The pop3Transport object is initialized.
- **Called by:** mailbox
- **Calls:** None

- **Method Name:** `returnCode pop3Transport::readHeaders(vector<wstring>& headers)`
- **Parameters:** headers. A reference to a vector<wstring> object, used to store e-mail headers.
- **Return Value:** One of the following:
  - NOERROR The method succeeded
  - MAIL_POPUNAVAIL The pop3 server was unavailable
  - MAIL_POPREFUSE The pop3 sever refused connection
- **Description:** Reads headers on a pop3 server.
- **Pre-conditions:** The parameter is a empty vector.
- **Post-conditions:** Headers are downloaded from the server and added to the vector.
- **Called by:** mailbox
- **Calls:** Functions in the e-mail library.

- **Method Name:** `returnCode pop3Transport::download(pop3messageIdentifier messageId, mail& message)`
- **Parameters:**
  - message id. An identifier to identify a message on a pop3 server. The exact format for this is not decided yet
  - message. A reference to a mail object.
- **Return Value:** One of the following:
  - NOERROR The method succeeded
  - MAIL_POPUNAVAIL The pop3 server was unavailable
  - MAIL_POPREFUSE The pop3 sever refused connection
- **Description:** Downloads a specified message from a pop3 server
- **Pre-conditions:** None
- **Post-conditions:** The e-mail is downloaded and stored in message
- **Called by:** mailbox
- **Calls:** Functions in the e-mail library

External communication and RES communication components

- **Method Name:** `returnCode externalCom::exportGrades(int sysID)`
- **Parameters:**
  - sysID. The internal identifier for a defined external system.
- **Return Value:** One of the following:
  - NOERROR The method succeeded
  - EXTSYSCOMP_NOFOUND Impossible to find the specified external communication component
  - EXTSYS_ERR Impossible to communicate with the external system
- **Description:** Exports the grades of the assignments of the currently opened course to the specified external system. This method implements the user requirement 6.2.1
- **Data structure and tables it accesses:**
  - ExternalSystemsInfoData from GlobalData (to define)
- **Pre-conditions:** Accurate data loaded into storage objects
- **Post-conditions:** The grades for the existent assignments in the course are exported to the specified external system
- **Called by:** Main application.
- **Calls:** Functions in the storage objects, externalCom::createGradesExcFile(), execution of RESCom::importGrades()

- **Method Name:** `returnCode externalCom::importStudents(int sysID)`
- **Parameters:**
  - sysID. The internal identifier for a defined external system.
- **Return Value:** One of the following:
  - NOERROR The method succeeded
  - EXTSYSCOMP_NOFOUND Impossible to find the specified external communication component
  - EXTSYS_ERR Impossible to communicate with the external system
- **Description:** Imports the students from the specified external system into the currently opened course. This method implements the user requirement 6.1.1
- **Data structure and tables it accesses:**
  - ExternalSystemsInfoData from GlobalData (to define)
- **Pre-conditions:** Accurate data loaded into storage objects
- **Post-conditions:** The students from the specified external system are loaded into the currently opened course.

- **Called by:** Main application.
- **Calls:** Functions in the storage objects, externalCom::loadStudentsExcFile(), execution of RESCom::exportStudents(std::wstring m_sCourseCode)

- **Method Name:** `returnCode externalCom::createGradesExcFile()`
- **Return Value:** One of the following:
  - NOERROR The method succeeded
  - ERROR Impossible to create the grades exchange file
- **Description:** Creates the exchange grades file used by the specified external system communication component. This method will implement the user requirement 6.2.2
- **Data structure and tables it accesses:**
  - AssignmentInfoData from courseData
  - StudentData from courseData
  - AssignmentData from studentData
  - ExternalSystemsInfoData from GlobalData (to define)
- **Pre-conditions:** Accurate data loaded into storage objects
- **Post-conditions:** An exchange file with the grades for the existent assignments in the course is created.
- **Called by:** externalCom::exportGrades(int sysID)
- **Calls:** Functions in the storage objects.

- **Method Name:** `returnCode externalCom::loadStudentsExcFile()`
- **Return Value:** One of the following:
  - NOERROR The method succeeded
  - ERROR Impossible to read the students exchange file
- **Description:** Imports the students from an exchange file into the currently opened course.
- **Data structure and tables it accesses:**
  - StudentData from courseData
  - CourseDataStorage
- **Pre-conditions:** Accurate data loaded into storage objects
- **Post-conditions:** The students from the exchange file are loaded into the currently opened course.
- **Called by:** externalCom::loadStudentsExcFile()
- **Calls:** Functions in the storage objects

NOTE: The two next methods belong to two independent single-class applications (scripts) that complement the AAMS system. In order to keep consistent with the earlier sections and the used notation, they appear as methods of an artificial class called *RESCom*.

- **Method Name:** `void RESCom::importGrades()`
- **Description:** Imports the grades from the exchange file into the RES system. This method implements the user requirement 6.2.1
- **Data structure and tables it accesses:**
  - RES system (file "res")
- **Pre-conditions:** The information in the exchange file (course code, etc.) is consistent with the information in the RES system.
- **Post-conditions:** The grades from the exchange file are imported to the RES system
- **Called by:** RES system importing script application.

- **Method Name:** `void RESCom::exportStudents(std::wstring m_sCourseCode)`
- **Parameters:**
  - m_sCourseCode. The code of the target course in the RES system
- **Description:** Exports a list with the students of the RES system course with code *m_sCourseCode*. This method implements the user requirement 6.1.1
- **Data structure and tables it accesses:**
  - RES system (file "res")
- **Pre-conditions:** The given code is a valid one in the RES system.
- **Post-conditions:** A students exchange file with the RES system students is created.
- **Called by:** RES system exporting script application.

GUI/Event handler component

- **Method Name:** `bool AAMSGUI::OnInit()`
- **Return Value:** One of the following:
  - AAMS_START The start is successful and the GUI can be displayed
  - AAMS_FAILURE Impossible to start the system
- **Description:** Initializes the main system objects and GUI elements and runs the system.
- **Data structure and tables it accesses:**
  - Defined GUI elements in the *GUIcreator* class.
- **Pre-conditions:** The operative system is ready and able to run the system
- **Post-conditions:** The system is initialized and the main window is shown
- **Called by:** The operative system ("main()" method).
- **Calls:** Initializing elements of the GUI library defined by the class *GUIcreator*.

The class *GUIcreator* contains the definition of the whole AAMS GUI. For each form, a wxWidgets class is defined. The controls inside these are defined as attributes, and each wxWidgets class contains a constructor method, where the static properties of every control are defined, and an event table in order to route the events to the *GUIcontroller* object. These methods are trivial and for clarity reasons they will not be listed here.

The class *GUIcontroller* contains a set of methods that handle all the user events, calling the proper methods of every component, displaying the corresponding forms, and managing the possible errors.

- **Method Name:** `void ErrorManager::handleUsrError(int errorCode)`
- **Parameters:**
  - errorCode. Identifier of the kind of error.
- **Description:** Handles possible errors, displaying explanations of them or finishing the system if they are critical.
- **Post-conditions:** The error is managed and an explanation of it is shown.
- **Called by:** GUIController.
- **Calls:** WindowManager::displayMsgBoxErr(wstring msg)

- **Method Name:** `returnCode FileManager::openExternalFile(wstring filePath)`
- **Parameters:**

  - filePath. Path where the file to be opened is.
- **Return Value:** One of the following:
  - NOERROR The method succeeded
  - UNEXISTENT_FILE The file could not be found
  - UNEXISTENT_APP There is no declared external application to open the file
- **Description:** Opens the given file with an external application.
- **Pre-conditions:** The operative system keeps a list of default applications for different file extensions.
- **Post-conditions:** The error is managed and an explanation of it is shown.
- **Called by:** GUIController.
- **Calls:** wxMimeTypesManager::GetFileTypeFromExtension(extension), wxFileType::GetOpenCommand(fileName)

- **Method Name:** `void WindowManager::displayMsgBoxErr(wstring msg)`
- **Parameters:**
  - msg. The message to be displayed
- **Description:** Shows a typical user error message dialog with the given text
- **Post-conditions:** The message box with the specified message is shown
- **Called by:** GUIcontroller.
- **Calls:** wxMessageDialog::wxMessageDialog(), wxMessageDialog::ShowModal()

- **Method Name:** `void WindowManager::showSecondaryWindow(int wCode)`
- **Parameters:**
  - wCode. Code that identifies the secondary window that will be opened
- **Description:** Shows a secondary window, disabling the main one at the same time
- **Pre-conditions:** There is no secondary window already opened
- **Post-conditions:** The secondary window is displayed and the main window is disabled
- **Called by:** GUIcontroller.
- **Calls:** Show the target window, disable the main one

- **Method Name:** `void WindowManager::hideSecondaryWindow()`
- **Description:** Hides the currently displayed secondary window, giving the focus back to the main one at the same time
- **Pre-conditions:** There is a secondary window already opened

- **Post-conditions:** The secondary window is hidden and the main window is enabled
- **Called by:** GUIcontroller.
- **Calls:** Show the main window, hide the secondary one

Data storage component

In this subsection we will list the attributes and methods for each class before we go through them. The attributes will be prefixed by a - sign and the methods will be prefixed by a + sign.

Class EncryptUtil

```
+ void Encrypt(std::string &sData)
+ void Decrypt(std::string &sData)
```

- **Method Name:** `void Encrypt(std::string &sData)`
- **Description:** The method encrypt the data in parameter sData.
- **Parameters:**
  - sData: The data to be encrypted.
- **Pre-condition:** The data in sData is not encrypted.
- **Post-condition:** The data in sData is encrypted.
- **Called by:** XMLUnsentEmailsDataWrite::WriteXMLUnsentEmailsData(), XMLGlobalDataWrite::WriteXMLGlobalData and XMLCourseDataWrite::WriteXMLCourseData

- **Method Name:** `void Decrypt(std::string &sData)`
- **Description:** The method decrypt the data in parameter sData.
- **Parameters:**
  - sData: The data to be decrypted.
- **Pre-condition:** The data in sData is encrypted.
- **Post-condition:** The data in sData is not encrypted.
- **Called by:** XMLUnsentEmailsDataRead::ReadXMLUnsentEmailsData(), XMLGlobalDataRead::ReadXMLGlobalData and XMLCourseDataRead::ReadXMLCourseData

Class UnsentEmailsStorage

```
- int m_nEmailIdCounter
- std::list<UnsentEmailsData> m_unsentEmailsData
+ bool Load()
```

+ `bool Save()`

+ `void AddEmail(const std::wstring sDestination, const std::wstring sHeader, const std::wstring sContent)`

+ `void RemoveEmail(int eid)`

+ `const std::list<UnsentEmailsData> & GetUnsentEmails()`

+ `int GetEmailCount()`

- **Method Name:** `bool Load()`
- **Description:** The method loads the unsent emails from a file.
- **Return value:** true if the load from file succeeded, otherwise false.
- **Post-condition:** The data in the file is now in loaded into std::list<UnsentEmailsData>
- **Called by:** GUIController
- **Calls:** XMLUnsentEmailsDataRead::ReadXMLUnsentEmailsData(), XMLUnsentEmailsDataRead::Open() and XMLUnsentEmailsDataRead::Close().

- **Method Name:** `bool Save()`
- **Description:** The method saves the unsent emails to a file.
- **Return value:** true if the save to file succeeded, otherwise false.
- **Pre-condition:** None
- **Validity checks:** None
- **Post-condition:** The unsent e-mails are saved to a file.
- **Called by:** GUIController
- **Calls:** XMLUnsentEmailsDataWrite::WriteXMLUnsentEmailsData(), XMLUnsentEmails-DataWrite::Open(), XMLUnsentEmailsDataWrite::Close()

- **Method Name:** `void AddEmail(const std::wstring sDestination, const std::wstring sHeader, const std::wstring sContent)`
- **Description:** The method add the email with the data from the parameters into the list of unsent emails
- **Parameters:**
  - sDestionation: Contain the destination e-mail.
  - sHeader: Contain the subject line of the e-mail.
  - sContent: Contain the content of the e-mail.
- **Pre-condition:** None
- **Validity checks:** None

- **Post-condition:** The e-mail specified by the parameters exists in the list of unsent emails m_unsentEmailsData
- **Calls:** None

- **Method Name:** `void RemoveEmail(int eid)`
- **Description:** The method remove the unsent e-mail specified by the email identifier eid
- **Parameters:**
  - eid: An identifier for an unsent e-mail in the list of unsent e-mails m_unsentEmailsData.
- **Pre-condition:** None
- **Validity checks:**
- **Post-condition:** The e-mail with the identifier eid is no longer in the list of unsent e-mails m_unsentEmailsData
- **Calls:** None

- **Method Name:** `const std::list<UnsentEmailsData> & GetUnsentEmails()`
- **Description:** The method return the list of unsent e-mails, m_unsentEmailsData.
- **Return value:** The list of unsent e-mails, m_unsentEmailsData.
- **Pre-condition:** None
- **Validity checks:**
- **Post-condition:** The list of unsent e-mails m_unsentEmailsData remains intact.
- **Calls:** None

- **Method Name:** `int GetEmailCount()`
- **Description:** The method returns the number of unsent e-emails from the list of unsent e-mails m_unsentEmailsData.
- **Return value:** The number of unsent e-mails in the list of unsent e-mails.
- **Pre-condition:** None
- **Validity checks:**
- **Post-condition:** The list of unsent e-mails m_unsentEmailsData remains intact.
- **Calls:** None

Class UnsentEmailsData

- std::wstring m_sDestination

- std::wstring m_sHeader

- std::wstring m_sContent

Class XMLUnsentEmailsDataRead

+ bool Open()

+ void ReadXMLUnsentEmailsData(std::list<UnsentEmailsData> &unsentEmailsData, int &emailIdCounter)

+ void Close()

- TiXmlDocument m_doc

- **Method Name:** `bool Open()`
- **Description:** The method opens the XML file containing the unsent emails for reading.
- **Return value:** true if opening of file succeeded, otherwise false.
- **Pre-condition:** None
- **Validity checks:**
- **Post-condition:** Variable m_doc contain the XML file.
- **Called by:** UnsentEmailsStorage::Load()
- **Calls:** None

- **Method Name:** `void ReadXMLUnsentEmailsData(std::list<UnsentEmailsData> &unsentEmailsData, int &emailIdCounter)`
- **Description:** The method read the XML value
- **Parameters:**
  - unsentEmailsData: The list of unsent emails.
  - emailIdCounter: The identification counter.
- **Return value:** None
- **Pre-condition:** The XML file have been successfully opened.
- **Validity checks:** If the file is opened.
- **Post-condition:** the parameters contain the data that was in the XML file.
- **Called by:** UnsentEmailsStorage::Load()
- **Calls:** None

- **Method Name:** `void Close()`
- **Description:** The method will close the file if opened.
- **Return value:** None
- **Pre-condition:** None
- **Validity checks:** Will check if file is opened.
- **Post-condition:** The file is closed.
- **Called by:** UnsentEmailsStorage::Load()
- **Calls:** None

Class XMLUnsentEmailsDataWrite

```
+ bool Open()
+ void WriteXMLUnsentEmailsData(const std::list<UnsentEmailsData> &unsentEmailsData,
  int emailIdCounter)
+ void Close()
- TiXmlDocument m_doc
```

- **Method Name:** `bool Open()`
- **Description:** The method will open the XML file for writing.
- **Return value:** true if the file is successfully opened, otherwise false.
- **Pre-condition:** None
- **Validity checks:** None
- **Post-condition:** None
- **Called by:** UnsentEmailsStorage::Save()
- **Calls:** None

- **Method Name:** `void WriteXMLUnsentEmailsData(const std::list<UnsentEmailsData> &unsentEmailsData, int emailIdCounter)`
- **Description:** The method will write the content of the parameters unsentEmailsData and emailIdCounter into the XML file that hold the unsent emails.
- **Return value:** None
- **Pre-condition:** None
- **Validity checks:** Will check if the file is opened.

- **Post-condition:** The data is written to the file.
- **Called by:** UnsentEmailsStorage::Save()
- **Calls:** None

- **Method Name:** `void Close()`
- **Description:** The method close the file if it is opened.
- **Return value:** None
- **Pre-condition:** None
- **Validity checks:** Will check if the file is opened.
- **Post-condition:** The file is closed
- **Called by:** UnsentEmailsStorage::Save()
- **Calls:** None

Class GlobalStorage

```
+ bool Load()
+ bool Save()
+ void SetSMTPUser(const std::string sUser)
+ void SetSMTPPass(const std::string sPass)
+ void SetSMTPAdress(const std::string sAdress)
+ void SetSMTPPort(const int nPort)
+ const GlobalData &GetGlobalData()
- GlobalData m_globalData
```

- **Method Name:** `bool Load()`
- **Description:** The method loads the global data from file
- **Return value:** true if the load from file succeeded otherwise false
- **Pre-condition:**
- **Validity checks:**
- **Post-condition:** The data from the file is loaded into m_globalData
- **Called by:** GUIController
- **Calls:** XMLGlobalDataRead::Open() , XMLGlobalDataRead::Close() and XMLGlobalDataRead::ReadXMLGlobalData()

- **Method Name:** `bool Save()`
- **Description:** The method saves the data in m_globalData to a file.
- **Return value:** None
- **Pre-condition:**
- **Validity checks:**
- **Post-condition:** The data in m_globalData is saved to a file.
- **Called by:** XMLGlobalDataWrite::Open() , XMLGlobalDataWrite::WriteXMLGlobalData() and XMLGlobalDataWrite::Close()
- **Calls:** None

- **Method Name:** `void SetSMTPUser(const std::string sUser)`
- **Description:** The method sets the user name for the SMTP server.
- **Return value:** None
- **Pre-condition:** None
- **Validity checks:**
- **Post-condition:** The m_sSMTPUser of m_globalData has the user name specified by parameter sUser.
- **Called by:** GUIController
- **Calls:** None

- **Method Name:** `void SetSMTPPass(const std::string sPass)`
- **Description:** The method set the password for the SMTP server.
- **Return value:** None
- **Pre-condition:** None
- **Validity checks:** None
- **Post-condition:** The m_sSMTPPass field of m_globalData have the password specified by parameter sPass.
- **Called by:** GUIController
- **Calls:** None

- **Method Name:** `void SetSMTPAdress(const std::string sAdress)`
- **Description:** The method set the address for the SMTP server.
- **Return value:** None

- **Pre-condition:** None
- **Validity checks:** None
- **Post-condition:** The m_sSMTPAdress field of m_globalData have the address specified by parameter sAdress.
- **Called by:** GUIController
- **Calls:** None

- **Method Name:** `void SetSMTPPort(const int nPort)`
- **Description:** The method set the port for the SMTP server.
- **Return value:** None
- **Pre-condition:** None
- **Validity checks:** None
- **Post-condition:** The m_nSMTPPort field of m_globalData have the port specified by parameter nPort.
- **Called by:** GUIController
- **Calls:** None

- **Method Name:** `const GlobalData &GetGlobalData()`
- **Description:** The method return the m_globalData for reading.
- **Return value:** The data structure m_globalData
- **Pre-condition:** None
- **Validity checks:** None
- **Post-condition:** The data of m_globalData remains intact.
- **Called by:** Different components
- **Calls:** None

Class GlobalData

- `std::string m_sSMTPUser`
- `std::string m_sSMTPPass`
- `std::string m_sSMTPAdress`
- `int m_nSMTPPort`

Class XMLGlobalDataRead

```
+ bool Open()
+ void ReadXMLGlobalData(GlobalData &globalData)
+ void Close()
+ TiXmlDocument m_doc
```

- **Method Name:** `bool Open()`
- **Description:** The method opens the file for reading.
- **Return value:** true if file was successfully opened.
- **Pre-condition:** None
- **Validity checks:** None
- **Post-condition:** None
- **Called by:** GlobalStorage::Load()
- **Calls:** None

- **Method Name:** `void ReadXMLGlobalData(GlobalData &globalData)`
- **Description:** The method reads the file and stores the information from the file in globalData parameter.
- **Parameters:**
  - globalData: holds the global data.
- **Return value:** None
- **Pre-condition:** None
- **Validity checks:** Will check if the file is opened.
- **Post-condition:** The data from the file is in the parameter globalData.
- **Called by:** GlobalStorage::Load()
- **Calls:** None

- **Method Name:** `void Close()`
- **Description:** The method closes the file if it is opened.
- **Return value:** None
- **Pre-condition:** None

- **Validity checks:** Will check if the file is opened
- **Post-condition:** The file is closed.
- **Called by:** GlobalStorage::Load()
- **Calls:** None

Class XMLGlobalDataWrite

```
+ bool Open()
+ void WriteXMLGlobalData(const GlobalData &globalData)
+ void Close()
- TiXmlDocument m_doc
```

- **Method Name:** `bool Open()`
- **Description:** The method will open the file for writing.
- **Return value:** true if the file is opened successfully, otherwise false
- **Pre-condition:** None
- **Validity checks:** Will check if the file is opened.
- **Post-condition:** None
- **Called by:** GlobalStorage::Save()
- **Calls:** None

- **Method Name:** `void WriteXMLGlobalData(const GlobalData &globalData)`
- **Description:** The method will save the data in globalData to a file.
- **Parameters:**
  - globalData: contains the data to be written to file.
- **Return value:** None
- **Pre-condition:** None
- **Validity checks:** None
- **Post-condition:** None
- **Called by:** GlobalStorage::Save()
- **Calls:** None

- **Method Name:** `void Close()`
- **Description:** The method will close the file if it is opened.
- **Return value:** None
- **Pre-condition:** None
- **Validity checks:** Will check if the file is opened.
- **Post-condition:** The file is closed.
- **Called by:** GlobalStorage::Save()
- **Calls:** None

Class CourseDataStorage

```
+ bool LoadCourse(const string sName)
+ bool SaveCourse()
+ void NewCourse(const std::string sName)
+ void SetCourseName(const std::wstring sCourseName)
+ void SetCourseCode(const std::wstring sCourseCode)
+ void SetStudentName(const int sid, const std::wstring sName)
+ void SetStudentExternalId(const int sid, const std::wstring sExternalId)
+ void SetStudentEmail(const int sid, const std::wstring sEmail)
+ void SetStudentGrade(const int sid, const std::wstring sGrade)
+ void SetAssignmentName(const int aid, const std::wstring sName)
+ void SetAssignmentExternalId(const int aid, const std::string sName)
+ void SetAssignmentEmailHeader(const int aid, const std::wstring sEmailHeader)
+ void SetAssignmentGrade(const int sid, const int aid, const std::wstring sGrade)
+ void SetAssignmentComment(const int sid, const int aid, const std::wstring sComment)
+ void SetAssignmentModified(const int sid, const int aid)
+ void SetFileAssignment(const std::string sStudentEmail, const std::wstring sEmailHeader,
  const std::wstring sFilename)
+ void SetFileComment(const int sid, const int aid, const int fid, const std::wstring
  sComment)
+ void SetFileFilename(const int sid, const int aid, const int fid, const std::wstring
  sFileName)
+ void RemoveStudent(const int sid)
+ void RemoveAssignment(const int aid)
```

```
+ void RemoveFile(const int fid, const int aid)

+ void AddNewStudent(const std::wstring sName, const std::wstring sExternalId, const
  std::wstring sEmail)

+ void AddNewAssignment(const std::wstring sAssignmentName, const std::wstring sExternalId,
  const std::wstring sEmailHeader)

+ void AddNewFile(const int aid, const std::wstring &sExtension)

+ const CourseData &GetCourseData()

+ const EmailAccountPOP3Data &GetEmailPOP3()

- CourseData m_courseData

- int m_nStudentIdCounter

- int m_nAssignmentIdCounter

- int m_nFileIdCounter

- EmailAccountPOP3Data m_POP3
```

- **Method Name:** `bool LoadCourse(const string sName)`
- **Description:** The method will load a course with the name specified by the parameter.
- **Return value:** true if the loading of the course was successful.
- **Pre-condition:** None
- **Validity checks:** Will check if the load was successful.
- **Post-condition:** None
- **Called by:** GUIController
- **Calls:** XMLCourseDataRead::Open() , XMLCourseDataRead::ReadXMLCourseData() and XMLCourseDataRead::Close()

- **Method Name:** `bool SaveCourse()`
- **Description:** The method will save the course data to file.
- **Return value:** true if saving was successful, otherwise false.
- **Pre-condition:** None
- **Validity checks:** Will check if the saving of the course was successful.
- **Post-condition:** None
- **Called by:** GUIController
- **Calls:** XMLCourseDataWrite::Open() , XMLCourseDataWrite::WriteXMLCourseData() and XMLCourseDataWrite::Close()

- **Method Name:** `void NewCourse(const std::string sName)`
- **Description:** The method will create a new course. This method is the implementation of user requirements: 1.1.1.
- **Parameters:**
  - sName: the name of the course.
- **Return value:** None
- **Pre-condition:** None
- **Validity checks:** Will check if the course already exists.
- **Post-condition:** The course with the name is created.
- **Called by:** GUIController
- **Calls:** None

- **Method Name:** `void SetCourseName(const std::wstring sCourseName)`
- **Parameters:**
  - sCourseName: the name of an existing course.
- **Return value:** None
- **Pre-condition:** None
- **Validity checks:** None
- **Post-condition:** The data of m_courseData is updated with the data from the parameter.
- **Called by:** GUIController
- **Calls:** None

- **Method Name:** `void SetCourseCode(const std::wstring sCourseCode)`
- **Parameters:**
  - sCourseCode: A code of the course.
- **Return value:** None
- **Pre-condition:** None
- **Validity checks:** None
- **Post-condition:** The data of m_courseData is updated with the data from the parameter.
- **Called by:** GUIController
- **Calls:** None

- **Method Name:** `void SetStudentName(const int sid, const std::wstring sName)`

- **Parameters:**
  - sid: student identifier
  - sName: the name of the student with id sid.
- **Return value:** None
- **Pre-condition:** None
- **Validity checks:** None
- **Post-condition:** The data of m_courseData is updated with the data from the parameter.
- **Called by:** GUIController
- **Calls:** None

- **Method Name:** `void SetStudentExternalId(const int sid, const std::wstring sExternalId)`
- **Parameters:**
  - sid: student Identifier
  - sExternalId: the external identifier of a student.
- **Return value:** None
- **Pre-condition:** None
- **Validity checks:** None
- **Post-condition:** The data of m_courseData is updated with the data from the parameter.
- **Called by:** GUIController
- **Calls:** None

- **Method Name:** `void SetStudentEmail(const int sid, const std::wstring sEmail)`
- **Parameters:**
  - sid: student identifier
  - sEmail: the email of the student.
- **Return value:** None
- **Pre-condition:** None
- **Validity checks:** None
- **Post-condition:** The data of m_courseData is updated with the data from the parameter.
- **Called by:** GUIController
- **Calls:** None

- **Method Name:** `void SetStudentGrade(const int sid, const std::wstring sGrade)`

- **Parameters:**
  - sid: student identifier
  - sGrade: the grade of the student.
- **Return value:** None
- **Pre-condition:** None
- **Validity checks:** None
- **Post-condition:** The data of m_courseData is updated with the data from the parameter.
- **Called by:** GUIController
- **Calls:** None

- **Method Name:** `void SetAssignmentName(const int aid, const std::wstring sName)`
- **Parameters:**
  - aid: Assignment identifier.
  - sName: The name of the assignment.
- **Return value:** None
- **Pre-condition:** None
- **Validity checks:** None
- **Post-condition:** The data of m_courseData is updated with the data from the parameter.
- **Called by:** GUIController
- **Calls: None**

- void SetAssignmentExternalId(const int aid, const std::string sName)
- **Parameters:**
  - aid: Assignment identifier.
  - sName: The name of the assignment.
- **Return value:** None
- **Pre-condition:** None
- **Validity checks:** None
- **Post-condition:** The data of m_courseData is updated with the data from the parameter.
- **Called by:** GUIController
- **Calls:** None

- **Method Name:** `void SetAssignmentEmailHeader(const int aid, const std::wstring sEmailHeader)`

- **Parameters:**
  - aid: Assignment identifier.
  - sEmailHeader: The email header of the assignment.
- **Return value:** None
- **Pre-condition:** None
- **Validity checks:** None
- **Post-condition:** The data of m_courseData is updated with the data from the parameter.
- **Called by:** GUIController
- **Calls:** None

- **Method Name:** `void SetAssignmentGrade(const int sid, const int aid, const std::wstring sGrade)`
- **Description:** This method is the implementation of user requirements: 3.3.1.
- **Parameters:**
  - sid: StudentIdentifier.
  - aid: Assignment identifier.
  - sGrade: The grade of an assignment for the student.
- **Return value:** None
- **Pre-condition:** None
- **Validity checks:** None
- **Post-condition:** The data of m_courseData is updated with the data from the parameter.
- **Called by:** GUIController
- **Calls:** None

- **Method Name:** `void SetAssignmentComment(const int sid, const int aid, const std::wstring sComment)`
- **Parameters:**
  - sid: Student identifier.
  - aid: Assignment identifier.
  - sComment: The comment of the assignment for the student.
- **Return value:** None
- **Pre-condition:** None
- **Validity checks:** None
- **Post-condition:** The data of m_courseData is updated with the data from the parameter.

- **Called by:** GUIController
- **Calls:** None

- **Method Name:** `void SetAssignmentModified(const int sid, const int aid)`
- **Parameters:**
  - sid: Student identifier.
  - aid: Assignment identifier.
- **Return value:** None
- **Pre-condition:** None
- **Validity checks:** None
- **Post-condition:** The data of m_courseData is updated with the data from the parameter.
- **Called by:** GUIController
- **Calls:** None

- **Method Name:** `void SetFileAssignment(const std::string sStudentEmail, const std::wstring sEmailHeader, const std::wstring sFilename)`
- **Parameters:**
  - sStudentEmail: The email of the student
  - sEmailHeader: The header of the email.
  - sFilename: The name of the file.
- **Return value:** None
- **Pre-condition:** None
- **Validity checks:** None
- **Post-condition:** The data of m_courseData is updated with the data from the parameter.
- **Called by:** GUIController
- **Calls:** None

- **Method Name:** `void SetFileComment(const int sid, const int aid, const int fid, const std::wstring sComment)`
- **Description:** This method is the implementation of user requirements: 3.2.1
- **Parameters:**
  - sid: Student identifier
  - fid: File identifier
  - sComment: The comment of the file.

- **Return value:** None
- **Pre-condition:** None
- **Validity checks:** None
- **Post-condition:** The data of m_courseData is updated with the data from the parameter.
- **Called by:** GUIController
- **Calls:** None

- **Method Name:** `void SetFileFilename(const int sid, const int aid, const int fid, const std::wstring sFileName)`
- **Parameters:**
  - sid: Student identifier
  - fid: File identifier
  - sFilename: The name of the file.
- **Return value:** None
- **Pre-condition:** None
- **Validity checks:** None
- **Post-condition:** The data of m_courseData is updated with the data from the parameter.
- **Called by:** GUIController
- **Calls:** None

- **Method Name:** `void RemoveStudent(const int sid)`
- **Parameters:**
  - sid: Student identifier.
- **Return value:** None
- **Pre-condition:** None
- **Validity checks:** None
- **Post-condition:** The student with id from parameter sid is no longer inside m_courseData
- **Called by:** GUIController
- **Calls:** None

- **Method Name:** `void RemoveAssignment(const int aid)`
- **Description:** This method is the implementation of user requirements: 2.2.1.
- **Parameters:**

  - aid: Assignment identifier
- **Return value:** None
- **Pre-condition:** None
- **Validity checks:** None
- **Post-condition:** The assignment with id from parameter aid is no longer inside m_courseData
- **Called by:** GUIController
- **Calls:** None

- **Method Name:** `void RemoveFile(const int fid, const int aid)`
- **Description:** This method is the implementation of user requirements: 2.2.1.
- **Parameters:**
  - fid: File identifier
  - aid: Assignment identifier
- **Return value:** None
- **Pre-condition:** None
- **Validity checks:** None
- **Post-condition:** The file with id from parameter fid is no longer inside m_courseData.
- **Called by:** GUIController
- **Calls:** None

- **Method Name:** `void AddNewStudent(const std::wstring sName, const std::wstring sExternalId, const std::wstring sEmail)`
- **Description:** This method is the implementation of user requirements: 1.2.1.
- **Parameters:**
  - sName: The name of the student.
  - sExternalId: The external identifier of the student.
  - sEmail: The email of the student.
- **Return value:** None
- **Pre-condition:** None
- **Validity checks:** None
- **Post-condition:** The student with information from the parameters is now in m_courseData.
- **Called by:** GUIController
- **Calls:** None

- **Method Name:** `void AddNewAssignment(const std::wstring sAssignmentName, const std::wstring sExternalId, const std::wstring sEmailHeader)`
- **Description:** This method is the implementation of user requirements: 1.3.1.
- **Parameters:**
  - sAssignmentName: Name of the assignment.
  - sExternalId: External identifier of the assignment.
  - sEmailHeader: The header (subject) of the email.
- **Return value:** None
- **Pre-condition:** None
- **Validity checks:** None
- **Post-condition:** The assignment with information from parameters is now in m_courseData.
- **Called by:** GUIController
- **Calls:** None

- **Method Name:** `void AddNewFile(const int aid, const std::wstring &sExtension)`
- **Description:** This method is the implementation of user requirements: 2.1.1. and 2.1.4.
- **Parameters:**
  - aid: Assignment identifier
  - sExtension: Extension of the file.
- **Return value:** None
- **Pre-condition:** None
- **Validity checks:** None
- **Post-condition:** The file with information from parameters is now in m_courseData.
- **Called by:** GUIController
- **Calls:** None

- **Method Name:** `const CourseData &GetCourseData()`
- **Description:** This method is the implementation of user requirements: 5.1.1. 5.2.1. 5.3.1. and 6.2.1.
- **Return value:** A reference to m_courseData
- **Pre-condition:** None
- **Validity checks:** None
- **Post-condition:** The data of m_courseData remains intact.
- **Called by:** Different components

- **Calls:** None

- **Method Name:** `const EmailAccountPOP3Data &GetEmailPOP3()`
- **Return value:** A reference to m_POP3.
- **Pre-condition:** None
- **Validity checks:** None
- **Post-condition:** The data of m_POP3 remains intact.
- **Called by:** Different components
- **Calls:** None

Class XMLCourseDataRead

```
+ bool Open()
+ void ReadXMLCourseData(CourseData &courseData, EmailAccountPOP3Data &emailPOP3,
  EmailAccountSMTPData &emailSMTP, int &nStudentIdCounter, int &nAssignmentIdCounter,
  int &nFileIdCounter)
+ void Close()
- TiXmlDocument m_doc
```

- **Method Name:** `bool Open()`
- **Description:** The method will open the file for reading.
- **Return value:** true if the opening of the file was successful
- **Pre-condition:** None
- **Validity checks:** Will check if the open of file was successful
- **Post-condition:** None
- **Called by:** CourseDataStorage::LoadCourse()
- **Calls:** None

- **Method Name:** `void ReadXMLCourseData(CourseData &courseData, EmailAccountPOP3Data &emailPOP3, EmailAccountSMTPData &emailSMTP, int &nStudentIdCounter, int &nAssignmentIdCounter, int &nFileIdCounter)`
- **Description:** The method will read the file and store the data in the parameters.
- **Return value:** None

- **Pre-condition:** None
- **Validity checks:** Will check if the file is opened.
- **Post-condition:** None
- **Called by:** CourseDataStorage::LoadCourse()
- **Calls:** None

- **Method Name:** `void Close()`
- **Description:** The method will close the file.
- **Return value:** None
- **Pre-condition:** None
- **Validity checks:** Will check if the file is opened.
- **Post-condition:** The file is closed.
- **Called by:** CourseDataStorage::LoadCourse()
- **Calls:** None

Class XMLCourseDataWrite

```
+ bool Open()

+ void WriteXMLCourseData(const CourseData &courseData, const EmailAccountPOP3Data
  &emailPOP3, const EmailAccountSMTPData &emailSMTP, const int nStudentIdCounter,
  const int nAssignmentIdCounter, const int nFileIdCounter)

+ void Close()
```

- **Method Name:** `bool Open()`
- **Description:** The method will open the file for writing.
- **Return value:** true if the opening of the file was successful.
- **Pre-condition:** None
- **Validity checks:** Will check if the file was opened.
- **Post-condition:** None
- **Called by:** CourseDataStorage::SaveCourse()
- **Calls:** None

- **Method Name:** `void WriteXMLCourseData(const CourseData &courseData, const EmailAccountPOP3Data &emailPOP3, const EmailAccountSMTPData &emailSMTP, const int nStudentIdCounter, const int nAssignmentIdCounter, const int nFileIdCounter)`
- **Description:** The method will write the content of the parameters to a file.
- **Return value:** None
- **Pre-condition:** None
- **Validity checks:** Will check if the file is opened.
- **Post-condition:** None
- **Called by:** CourseDataStorage::SaveCourse()
- **Calls:** None

- **Method Name:** `void Close()`
- **Description:** The method will close the file.
- **Return value:** None
- **Pre-condition:** None
- **Validity checks:** Will check if the file is opened
- **Post-condition:** The file is closed.
- **Called by:** CourseDataStorage::SaveCourse()
- **Calls:** None

Class CourseData

- `std::wstring m_sCourseName`
- `std::wstring m_sCourseCode`
- `std::list<StudentData> m_StudentData`
- `std::list<AssignmentInfoData> m_assignmentInfoData`

Class StudentData

- `int m_sid`
- `std::wstring m_sName`
- `std::wstring m_sExternalId`

- std::wstring m_sEmail
- std::wstring m_sGrade
- std::list<AssignmentData> m_assignmentData

Class AssignmentData

- int m_aid
- std::wstring m_sComment
- std::wstring m_sGrade
- bool m_bModified
- std::list<FileData> m_fileData

Class FileData

- int m_fid
- std::wstring m_sFilename

Class AssignmentInfoData

- int m_aid
- std::wstring m_sName
- std::wstring m_sExternalId
- std::wstring m_sEmailHeader
- int m_nFileCount
- std::list<FileInfoData> m_fileInfoData

Class FileInfoData

- int m_fid
- std::wstring m_sExtension

Class EmailAccountPOP3Data

- std::string m_sUser
- std::string m_sPass
- std::string m_sAdress
- int m_nPort

Report generator component

- **Method Name:** `PDFGenerateType::getCourseData(std::wstring sCourseName)`
- **Return Value:** One of the following:
  - TRUE The method successfully executed
  - FALSE The method was unsuccessfully executed
- **Description:** Fetch data for a specific course.
- **Data structure and tables it accesses:**
  - Course data from courseData
  - Student data from courseData
  - Assignment data from courseData
- **Pre-conditions:** Course identifier
- **Post-conditions:** Relevant data to create a course specific report.
- **Called by:** GUIController
- **Calls:** None

- **Method Name:** `PDFGenerateType::getStudentData(std::wstring sStudentName)`
- **Return Value:** One of the following:
  - TRUE The method successfully executed
  - FALSE The method was unsuccessfully executed
- **Description:** Fetch data for a specific student.
- **Data structure and tables it accesses:**
  - course data from courseData
  - Student data from courseData
- **Pre-conditions:** Student identifier
- **Post-conditions:** Relevant data to create a student specific report.
- **Called by:** GUIController
- **Calls:** None

- **Method Name:** `PDFGenerateType::getStudentAssignmentData(std::wstring sAssName)`
- **Return Value:** One of the following:
  - TRUE The method successfully executed
  - FALSE The method was unsuccessfully executed
- **Description:** Fetch data for a specific assignment.

- **Data structure and tables it accesses:**
  - course data from courseData
  - Assignment data from courseData
- **Pre-conditions:** Assignment identifier
- **Post-conditions:** Relevant data to create a student specific report.
- **Called by:** GUIController
- **Calls:** None

- **Method Name:** `CreatePdf::mCourseReport()`
- **Return Value:** One of the following:
  - TRUE The method successfully executed
  - FALSE The method was unsuccessfully executed
- **Description:** Generate a report based on course data. This method is required for the implementation of the user requirement 5.1.1
- **Data structure and tables it accesses:**
  - course data from courseData
  - Student data from courseData
  - Assignment data from courseData
- **Pre-conditions:** Data to be formatted
- **Post-conditions:** Relevant data to create a course specific report.
- **Called by:** GUIController
- **Calls:** Haru Free PDF Library

- **Method Name:** `CreatePdf::mStudentReport()`
- **Return Value:** One of the following:
  - TRUE The method successfully executed
  - FALSE The method was unsuccessfully executed
- **Description:** Generate a report based on student data. This method is required for the implementation of the user requirement 5.2.1
- **Data structure and tables it accesses:**
  - course data from courseData
  - Student data from courseData
  - Assignment data from courseData
- **Pre-conditions:** Data to be formatted
- **Post-conditions:** Relevant data to create a student specific report.

- **Called by:** GUIController
- **Calls:** Haru Free PDF Library

- **Method Name:** `CreatePdf::mAssReport()`
- **Return Value:** One of the following:
  - TRUE The method successfully executed
  - FALSE The method was unsuccessfully executed
- **Description:** Generate a report based on student data. This method is required for the implementation of the user requirement 5.3.1
- **Data structure and tables it accesses:**
  - course data from courseData
  - Student data from courseData
  - Assignment data from courseData
- **Pre-conditions:** Data to be formatted
- **Post-conditions:** Data to create an assignment specific report.
- **Called by:** GUIController
- **Calls:** Haru Free PDF Library

Cross-reference index to the User Requirements

In this subsection we will list the functional user requirements specified in the section 4.a of [2], together with the classes that will provide each functionality. Each set of user requirements corresponds, approximately, to a different component of the system (e.g., the user requirement listed in *Reporting* are implemented by methods of classes that belong to the *Report generator* component).

Definition and management of the courses

- **1.1.1. Creation of a course:** class *CourseDataStorage* (*Data storage* component)
- **1.2.1. Importing students list:** class *externalCom* (*External communication* component), class *CourseDataStorage* (*Data storage* component)
- **1.3.1. Defining an assignment:** class *CourseDataStorage* (*Data storage* component)

Collection and organization of assignments

- **2.1.1. Input from students:** classes *mailbox* and *mail* (*Student communication* component)
- **2.1.2. Downloading assignments:** classes *mailbox*, *mail*, *validator* and *pop3Transport* (*Student communication* component)
- **2.1.3. Identifying e-mails:** class *validator* (*Student communication* component)
- **2.1.4. Manual handling of assignments:** class *CourseDataStorage* (*Data storage* component)
- **2.2.1. Deleting assignments:** class *CourseDataStorage* (*Data storage* component)
- **2.2.2. Organizing assignments:** class *GlobalStorage* (*Data storage* component)

Editing and grading of assignments

- **3.1.1. Opening files:** classes *FileManager* (*GUI/Event handler* component) and *CourseDataStorage* (*Data storage* component)
- **3.2.1. Commenting files:** class *CourseDataStorage* (*Data storage* component)
- **3.3.1. Grading assignments:** class *CourseDataStorage* (*Data storage* component)

Answer to the assignments

- **4.1.1. OK confirmation:** class *mailbox* (*Student communication* component)
- **4.1.2. Error notification:** classes *mailbox* and *smtpTransport* (*Student communication* component)
- **4.2.1. Sending feedback to students:** classes *mailbox* and *smtpTransport* (*Student communication* component)
- **4.2.2. Composition of feedback:** class *CourseDataStorage* (*Data storage* component) and classes *mailbox* and *smtpTransport* (*Student communication* component)

Reporting

- **5.1.1. Reports about the general state of a specific course:** classes *PDFGenerateType* and *CreatePdf* (*Report generator* component)
- **5.2.1. Reports about the state of a specific student:** classes *PDFGenerateType* and *CreatePdf* (*Report generator* component)
- **5.3.1. Reports about the state of a specific assignment:** classes *PDFGenerateType* and *CreatePdf* (*Report generator* component)

RES system link

- **6.1.1. List of students import:** class *RESCom* (*RES communication* component)
- **6.2.1. List of grades export:** class *RESCom* (*RES communication* component)

5.6. Package Diagram

- The *Data storage* package contain classes which handles all information flow and storage within AAMS and has dependencies such as the *GUI*, *External communication* and *Report generator* packages.
- The *Student communication* package contains classes to retrieve data from emails sent by students. The data storage classes depends on the data obtained by the *Student communication* classes.
- The *GUI* package contains classes to interact with *AAMSGUI* and functions, which depends on data from the *Data storage* package.
- The *External communication* package contains classes to communicate with the RES system. These classes use data which is obtained from the *Data storage* classes. The *External communication* classes are initiated by the *GUI* package.
- The *Report generator* package contains classes to generate reports based on data from the *Data storage* classes. The *Report generator* classes are initiated by the *GUI* package.

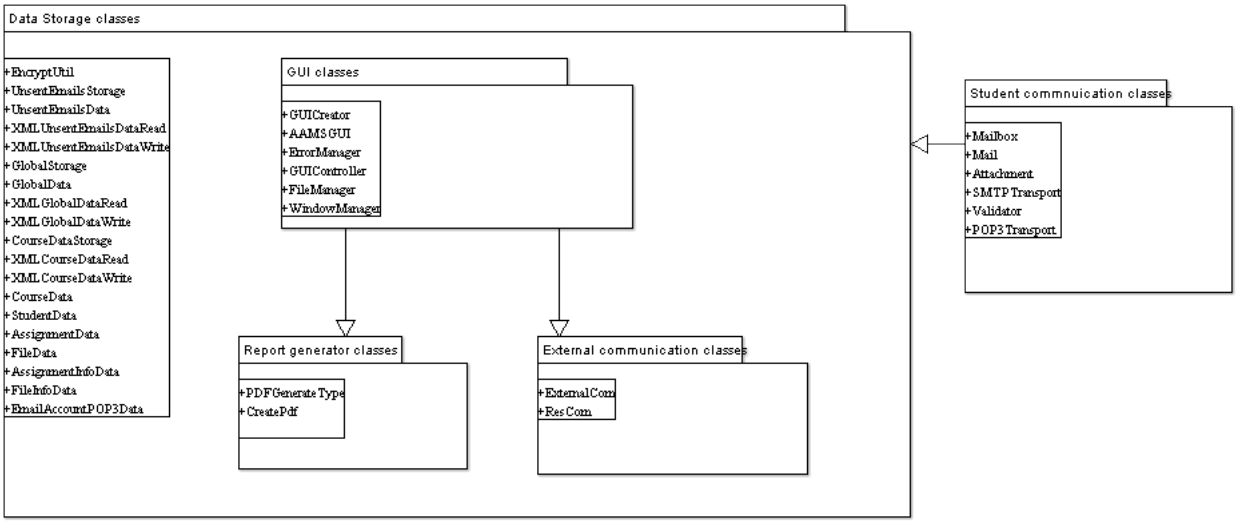


Figure 40: Package diagram of the system

6. Functional Test Cases

For testing the functionality of the system, we will provide a sequence of tests in which the tester will create a course, fill it with some information and extract some information from it. This sequence is intended to go through all the required functionalities of the system (see section 4.a in [2]). For keeping the test over the critical path, it is important, of course, to follow the sequence steps in the given order.

6.1. Test input data

All the test input data needed to perform the test sequence is listed below. Of course, the free text values, as the teacher's comment for an assignment, can be replaced for any others if it is considered preferable.

Global data

The specified SMTP server will probably not be reachable during a test situation due to restrictions in the local network. Use a reachable SMTP server.

- SMTP user: *professor_mvk21*
- SMTP pass: *sumvk07*
- SMTP server: *smtp.mail.yahoo.com*
- SMTP port: *465*
- ExtSysname: *RES*
- ExpName: *resExport*
- ImpName: *resImport*

Course data

- Course name: *Basic computer science*
- Course code: *test21mvk08*
- POP3 user: *professor_mvk21*
- POP3 pass: *professor_mvk21*
- POP3 address: *professor_mvk21@yahoo.se*
- POP3 server: *pop.mail.yahoo.com*
- POP3 port *995*

Student data

To use this account for sending and receiving mail, use the same settings as for the professors account except the user name that should be: *gordon.freeman29* .

- Student name: *Gordon Freeman*
- External id: (not used for tests with this student)
- E-mail: *gordon.freeman29@yahoo.se*

Assignment data

- Assignment name: *assignment 1*
- External id: *hem1*
- Expected email header: *ass1 08*
- Files: *.pdf, .odt*

6.2. Test sets

Starting and defining a course

General settings

- **Functionality description:** Test the functionality of the general settings
- **User requirement:** 4.2.1
- **Input:**
  - Login name for SMTP server
  - Password for SMTP server
  - Address for the SMTP server
  - Port for the SMTP server (optional, standard port is default)
- **Expected output:**
  - The information for logging in into the SMTP server is in the system
- **Test procedure:**
  - Start the system
  - Click the General options button
  - Enter login name
  - Enter password
  - Enter address
  - Enter port (optional)
  - Click OK button
  - Confirm that the introduced information is in the global database

Set up external system

- **Functionality description:** Test the functionality of connecting the system to new external system communication systems.
- **User requirement:** 6.1.1, 6.2.1
- **Input:**
  - The name of the executable for the external system to set up
- **Expected output:**
  - The name of the executable is now in the system
- **Test procedure:**
  - Start the system
  - Click the General options button
  - Click Add external system

- Input the name of the external executable
- Click OK button
- Click OK button
- Confirm that the introduced information is in the global database

Import students from the RES system

- **Functionality description:** Test the functionality of importing students from the RES system
- **User requirement:** 1.2.1. Importing students list
- **Input:**
  - Login name for SSH to the host my.nada.kth.se
  - Password for SSH to the host my.nada.kth.se
  - The course code.
- **Expected output:**
  - Each student in the course specified by the course code in RES is now imported in our system with their name, external identifier and e-mail.
- **Test procedure:**
  - Start the system
  - Load a course
  - Click the menu External
  - Click the menu item RES.
  - Click the menu item Import list of students
  - Enter login name
  - Enter password

Create and define a new course

- **Functionality description:** Test the functionality of defining a new course.
- **User requirement:** 1.1.1. Creation of a course
- **Input:**
  - Name of the course
  - Code for the course
- **Expected output:**
  - A new course is created in the system
  - A new file have been created for the new course
- **Test procedure:**
  - Start the system

- In the Course menu, click New Course
- In the empty input fields, input course name and course code

Create and define a student

- **Functionality description:** Test the functionality of defining a new student.
- **User requirement:** 1.1.1
- **Input:**
  - Name of the student
  - E-mail of the student
  - External identifier (optional)
  - Grade of the student (optional)
- **Expected output:**
  - A new student with the properties of the input has been created in the system for the current course.
- **Test procedure:**
  - Start the system
  - Open a course
  - Press the button for Adding a student.
  - Enter at least the non-optional input fields in the dialog that pop up.
  - Click OK button

Create and define an assignment

- **Functionality description:** Test the functionality of creating/defining an assignment.
- **User requirement:** 1.3.1. Defining an assignment
- **Input:**
  - A name for the assignment.
  - Email header (subject line for recognizing incoming e-mails to assignment)
  - External identifier (optional)
  - End date for assignment (optional)
- **Expected output:**
  - A new assignment with the properties of the input have been created in the system for the current course.
- **Test procedure:**
  - Start the system
  - Load a course
  - Press the button for creating a new assignment

- Enter at least the non-optional input fields in the dialog that pop up.
- Click OK button

Create and define a new file for an assignment

- **Functionality description:** Test the functionality for creating and defining a new file for an assignment.
- **User requirement:** 1.3.1
- **Input:**
  - The expected extension of the file.
- **Expected output:**
  - The file have been created and defined for an assignment for the current course in the system.
- **Test procedure:**
  - Start the system
  - Load a course
  - Select an assignment
  - Press Add file button
  - Enter the expected extension.
  - Press OK.

Filling in a course with information

Fetch assignments from the e-mail account

- **Functionality description:** Test e-mail fetching functionality. This is more a description of a series of possible tests than a single test.
- **User requirement:** 2.1.1, 2.1.2, 2.1.3, 4.1.1 and 4.1.2
- **Input:**
  - Course and student data setup as in previous tests.
  - E-mails sent from gordon.freeman29@yahoo.se, or some random address not specified in course data. The e-mails should be composed according to the following criteria.
    * Subject line containing both, one or none of the following strings: "ass1 08" and "test21mvk08". Subject line may contain some random text. Substrings of the subject line can be in random order.
    * A random text message of length ≥ 0.
    * Some files may be attached to the e-mail. Testing should be done using: a pdf-file and an odt-file, only one of them or none of them. To every combination a test with additional files should be performed.
- **Expected output:**
  - For any e-mail sent from gordon.freeman29@yahoo.se that has a subject line containing "ass1 08" and "test21mvk08" as disjunct substrings. The message should be recognized and registered in the course database. Any other subject line should be ignored and the e-mail should be left untouched on the POP3 server.
  - For a registered e-mail any attachment should be registered in the course database.
  - For any recognized e-mail a confirmation e-mail should be sent to gordon.freeman29@yahoo.se.
  - For any recognized e-mail that does not have a pdf-file and an odt-file attached the confirmation e-mail should contain an error report pointing out what is missing.
- **Test procedure:**
  - Setup course and student data as in previous tests.
  - Compose an e-mail and send it to professor_mvk21@yahoo.se, using some random e-mail application.
  - Use yahoo mail web service to monitor the e-mail account.
  - When the message is in the account inbox. Select "Download" from the "E-mail" menu.
  - Confirm that the course database is consistent with expected output.
  - Monitor the e-mail account, that was used for sending the e-mail, for expected confirmation e-mail.
  - Confirm that a confirmation e-mail is consistent with the expected output.
  - To do another permutation of this test delete course and student data. Otherwise leave it for other tests.

Import an assignment manually

- **Functionality description:** Testing manual import of assignments. This is more a description of a series of possible tests than a single test.

- **User requirement:** 2.1.4
- **Input:**
  - Course and student data setup as in previous tests.
  - One or more files to import. Any file format.
- **Expected output:**
  - The files should be registered in the course database as belonging to the selected student and assignment.
- **Test procedure:**
  - Setup course and student data as in previous tests.
  - In main window select "Student" tab or "Assignment" tab.
  - If "student" tab was selected, select "Gordon Freeman" in "Student" tab and click "View assignments" button. If "Assignment" tab was selected, select "assignment 1" in "Assignment" tab and click "View students" button.
  - If "Student" tab was selected, select "assignment 1" in "View assignments" dialog and click "Import file" button. If "Assignment" tab was selected, select "Gordon Freeman" in "View students" dialog and click "Import file" button.
  - With file chooser dialog navigate to the directory with the files to import and select a file and click "OK" button.
  - Repeat the two previous step for all files.
  - Confirm that files have been added to the course database using "View assignments" dialog or "View students" dialog

Set comments and grades for an assignment

- **Functionality description:** Testing commenting and grading assignments. This is more a description of a series of possible tests than a single test.
- **User requirement:** 3.2.1 and 3.3.1
- **Input:**
  - Course and student data setup as in previous tests.
  - Some random text to be used as a comment for an assignment.
  - Some random text to be used as a grade for an assignment.
- **Expected output:**
  - A comment and a grade registered in the course database as belonging to the selected student and assignment.
- **Test procedure:**
  - Setup course and student data as in previous tests.
  - In main window select "Student" tab or "Assignment" tab.
  - If "student" tab was selected, select "Gordon Freeman" in "Student" tab and click "View assignments" button. If "Assignment" tab was selected, select "assignment 1" in "Assignment" tab and click "View students" button.

- If "Student" tab was selected, select "assignment 1" in "View assignments" dialog and click "Edit comment and grade" button. If "Assignment" tab was selected, select "Gordon Freeman" in "View students" dialog and click "Edit comment and grade" button.
- In the "Comment and grade" dialog. Enter some random text in the "Comment" input box and "Grade" input field. Click "OK" button.
- Confirm that the comment and grade have been saved to the course database by following the previous steps to display the "Comment and grade" dialog for the same student and assignment.

Send feedback to a student

- **Functionality description:** Testing sending of feedback. This is more a description of a series of possible tests than a single test.
- **User requirement:** 4.2.1 and 4.2.2
- **Input:**
  - Course and student data setup as in previous tests.
  - Some random text to be used as a comment for an assignment.
  - Some random text to be used as a grade for an assignment.
- **Expected output:**
  - An e-mail sent to a registered students e-mail address.
- **Test procedure:**
  - Setup course and student data as in previous tests.
  - In main window select "Student" tab or "Assignment" tab.
  - If "student" tab was selected, select "Gordon Freeman" in "Student" tab and click "View assignments" button. If "Assignment" tab was selected, select "assignment 1" in "Assignment" tab and click "View students" button.
  - If "Student" tab was selected, select "assignment 1" in "View assignments" dialog and click "Edit comment and grade" button. If "Assignment" tab was selected, select "Gordon Freeman" in "View students" dialog and click "Edit comment and grade" button.
  - In the "Comment and grade" dialog. Enter some random text in the "Comment" input box and "Grade" input field. Click the "Send feedback" button or the "OK" button.
  - If the "OK" button was clicked, select "Send" from the "E-mail" menu.
  - Monitor the e-mail account for the selected student and confirm that an e-mail is sent to this address.
  - Confirm that the e-mail contains the comment and grade that was entered in the "Comment and grade" dialog.

Extracting information from a course

Test PDF Report generation

- **Functionality description:** (Generate specified report in PDF file format)
- **User requirement:** 5.1.1 , 5.2.1 , 5.3.1
- **Input:**
  - Assignment
  - Student
  - Course
- **Expected output:**

  One of the following files, according to the selected type, is created and displayed with an external viewer:
  - <course>.pdf
  - <assignment course>.pdf
  - <student course>.pdf
- **Test procedure:**
  - Choose to generate a report in AAMS
  - Choose which type (Assignment, Student or Course)
  - Analyze the output generated by AAMS

Export grades to the RES system

- **Functionality description:** (Exporting grades to the RES system)
- **User requirement:** 6.2.1
- **Input:**
  - XML file with students and grades for current course
- **Expected output:**
  - A dialog box confirming the exportation
- **Test procedure:**
  - Choose to export to RES
  - Wait for output

References

[1] Sommerville, I., *Software Engineering*, Addison-Wesley. Eighth edition (2007)

[2] Group 21, *Requirements Document*, 2007