AAMS

(Automatic Assignment Management System)

Group 21

Ajanth Thangavelu

Roberto Castañeda Lozano

Tony Karlsson

Love Jädergård

Design document

Contents

2. System overview

2.2. Overall architecture description

The system will follow the repository model discussed in the course book, 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*.

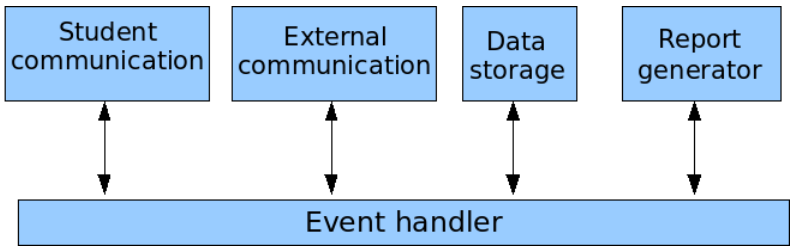

Figure 1: Control-flow model of the whole system

The Data storage component will be used as the main gate to retrieve and send data to every component, 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 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 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

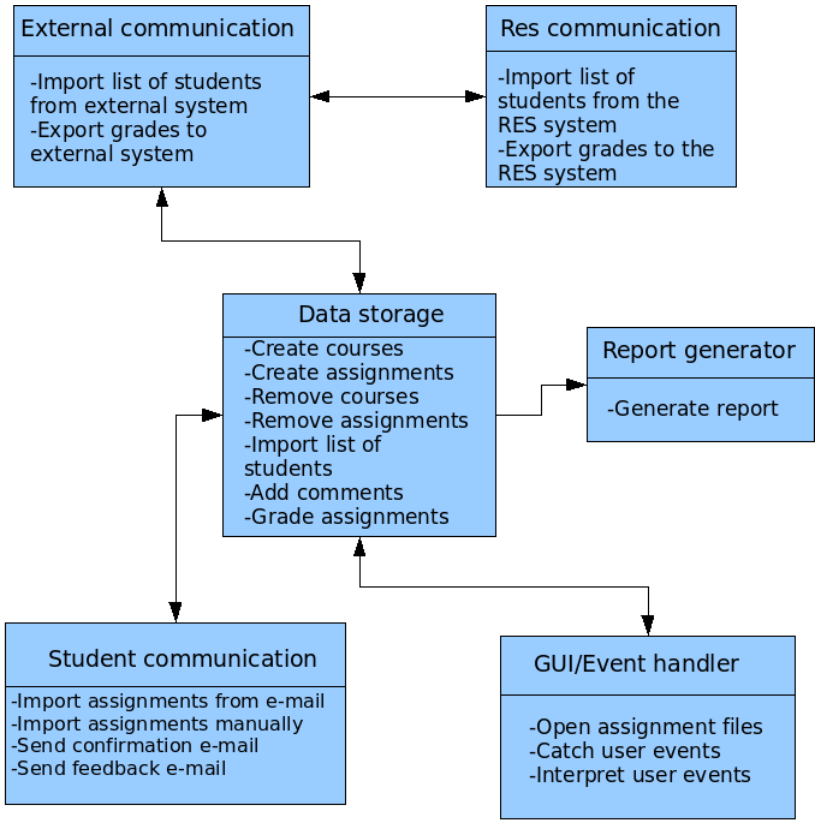

Figure 2: Data-flow model of the whole system

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. This layered structure 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

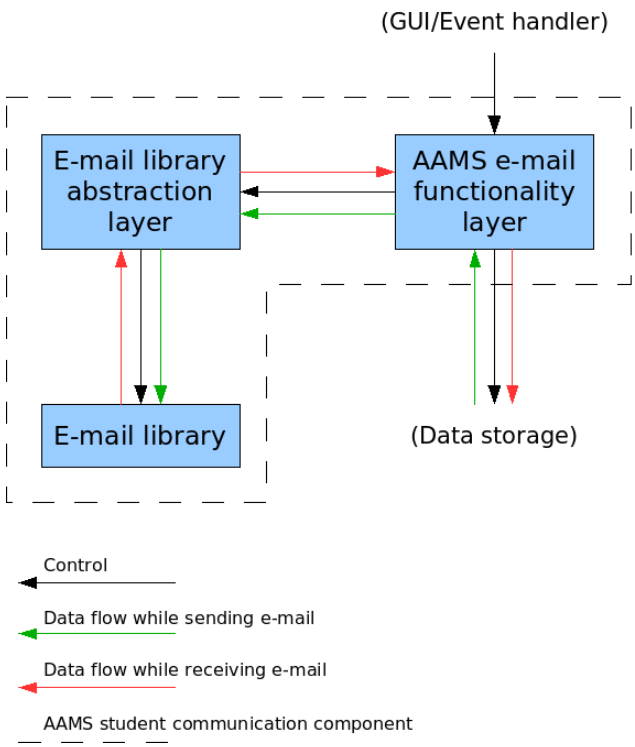

Figure 3: Control and data-flow model of the student communication component

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.

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.

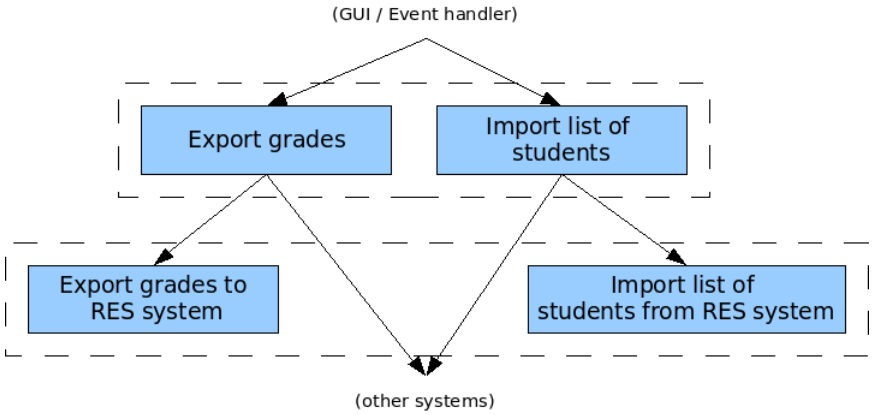

Figure 4: Control-flow model of the external communication and RES communication components

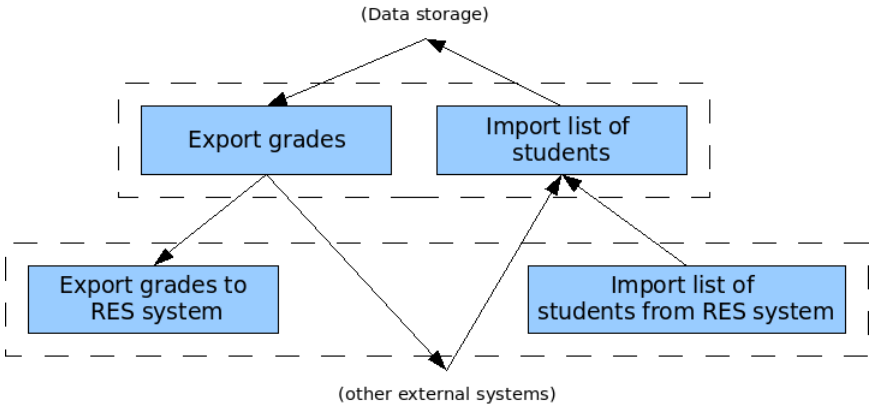

Figure 5: Data-flow model of the external communication and RES communication components

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.

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* frame-

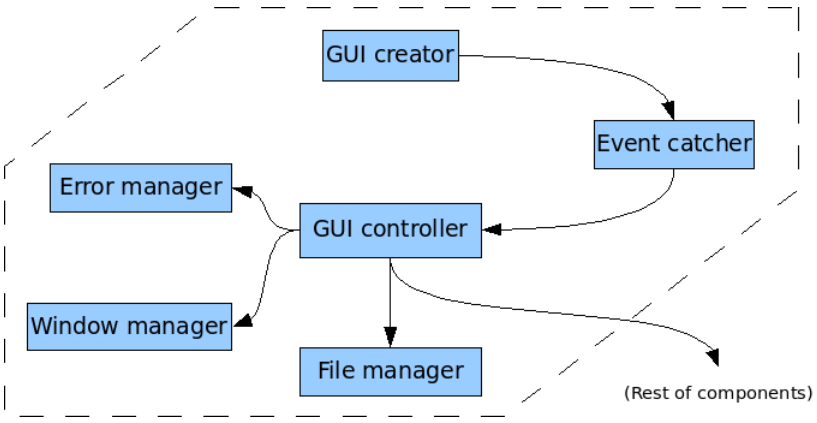

Figure 6: Control-flow model of the GUI/Event handler component

work, 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.

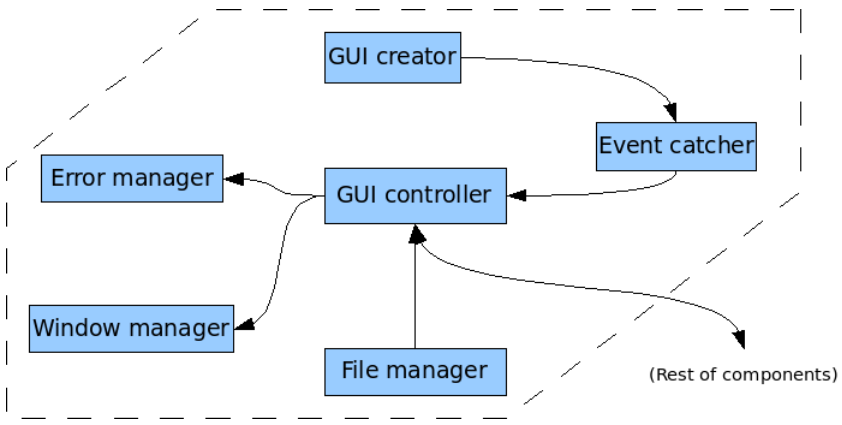

Figure 7: Data-flow model of the GUI/Event handler component

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

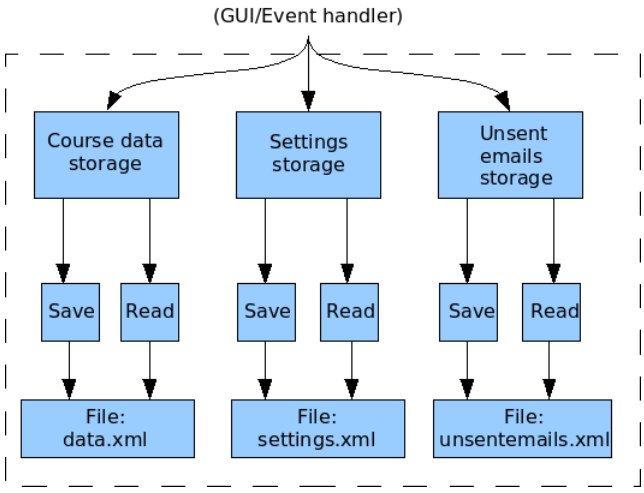

Figure 8: Control-flow model of the data storage component

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 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.

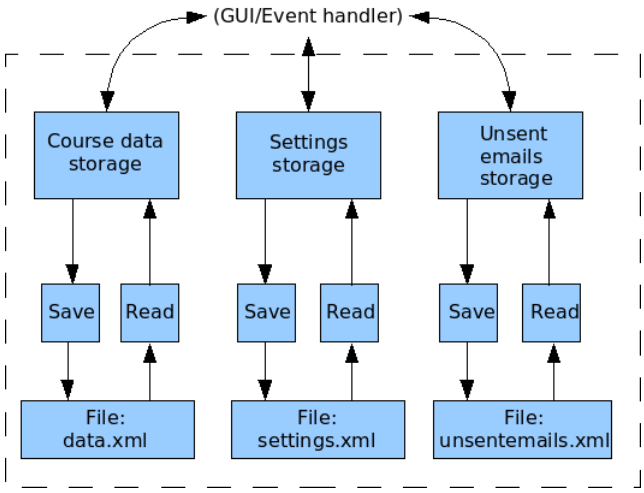

Figure 9: Data-flow model of the data storage component

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.

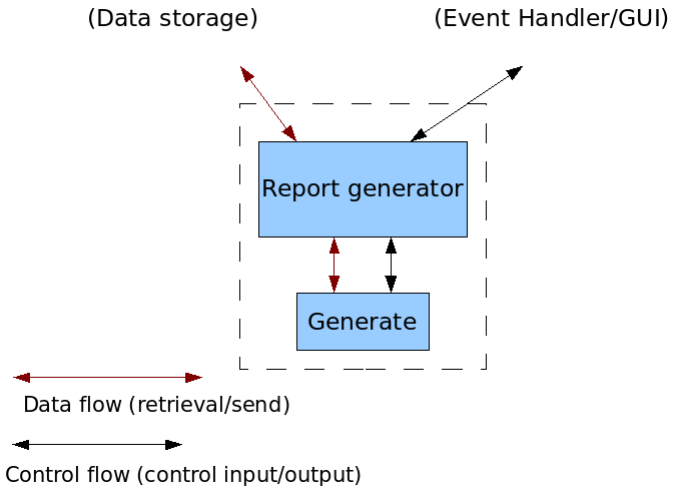

Figure 10: Control and data-flow model of the report generator component