Spreadsheet Reformatting Tool

Class Diagrams, System Architecture, and System Design

FHSU CSCI 441 Fall 2019

Group Members: John Mullane, Jeremy Pogue, Josh Lewis, Taylor Zwiebel

Project Website: https://sites.google.com/view/csci441vaf19-hrisreportmanagem
Github: https://github.com/jepogue/HRIS-Report-Management

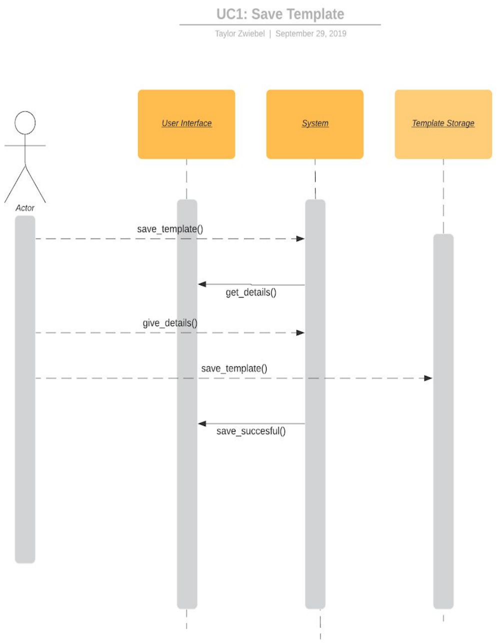
Individual Contributions Breakdown

Project Part		**Project Category**	**Team Member**			
			John	Jeremy	Taylor	Josh
			Mullane	Pogue	Zwiebel	Lewis
Responsibility %	1	Section 1 - interaction diagrams	33%		33%	33%
		References	100%			
		Project Management	33%	33%		33%
	2	2.a Class Diagram				100%
		2.b Data Types and Operation Signatures			100%	
		2.c Traceability Matrix		100%		
		3.a Architectural Styles	100%			
		3.b Identifying Subsystems	100%			
		3.c Persistent Data Storage	100%			
		3.d Global Flow Control	100%			
		3.e Hardware Requirements	100%			
		Project Management	33%		33%	33%

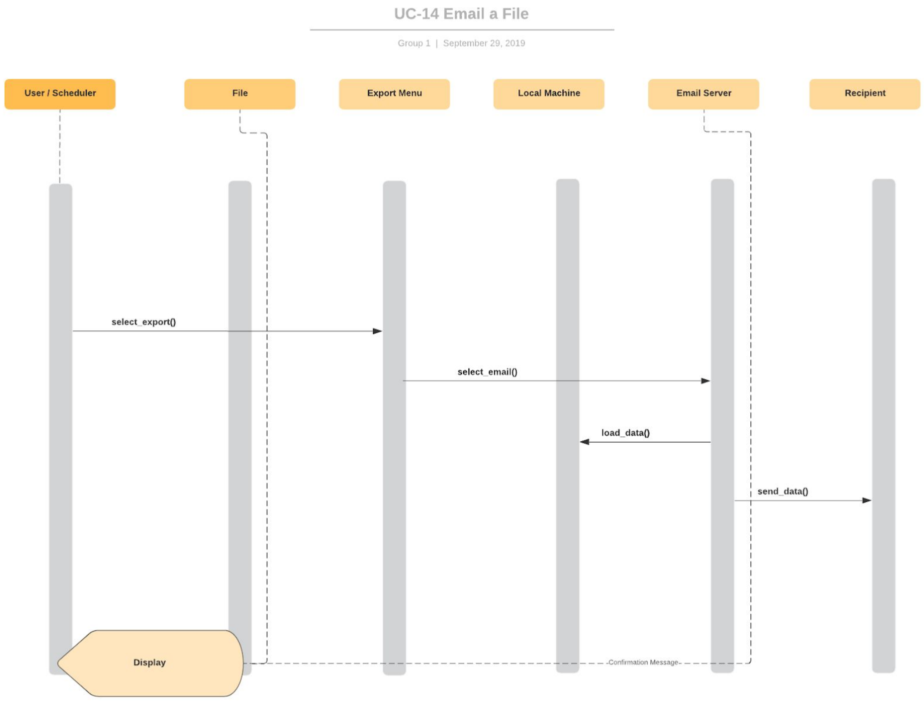
Contents

1. Interaction Diagrams

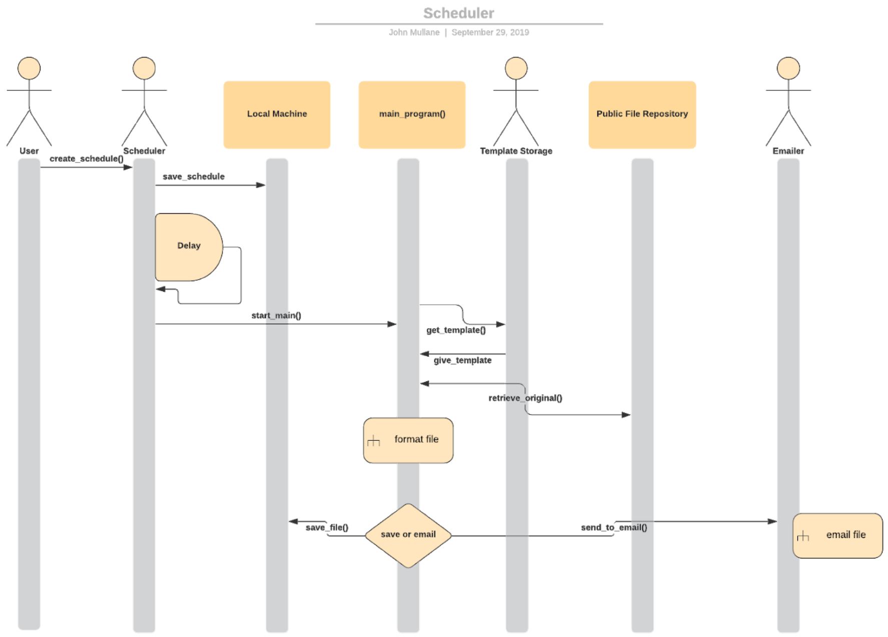
UC-1 Interaction Diagram



UC-14 Interaction Diagram

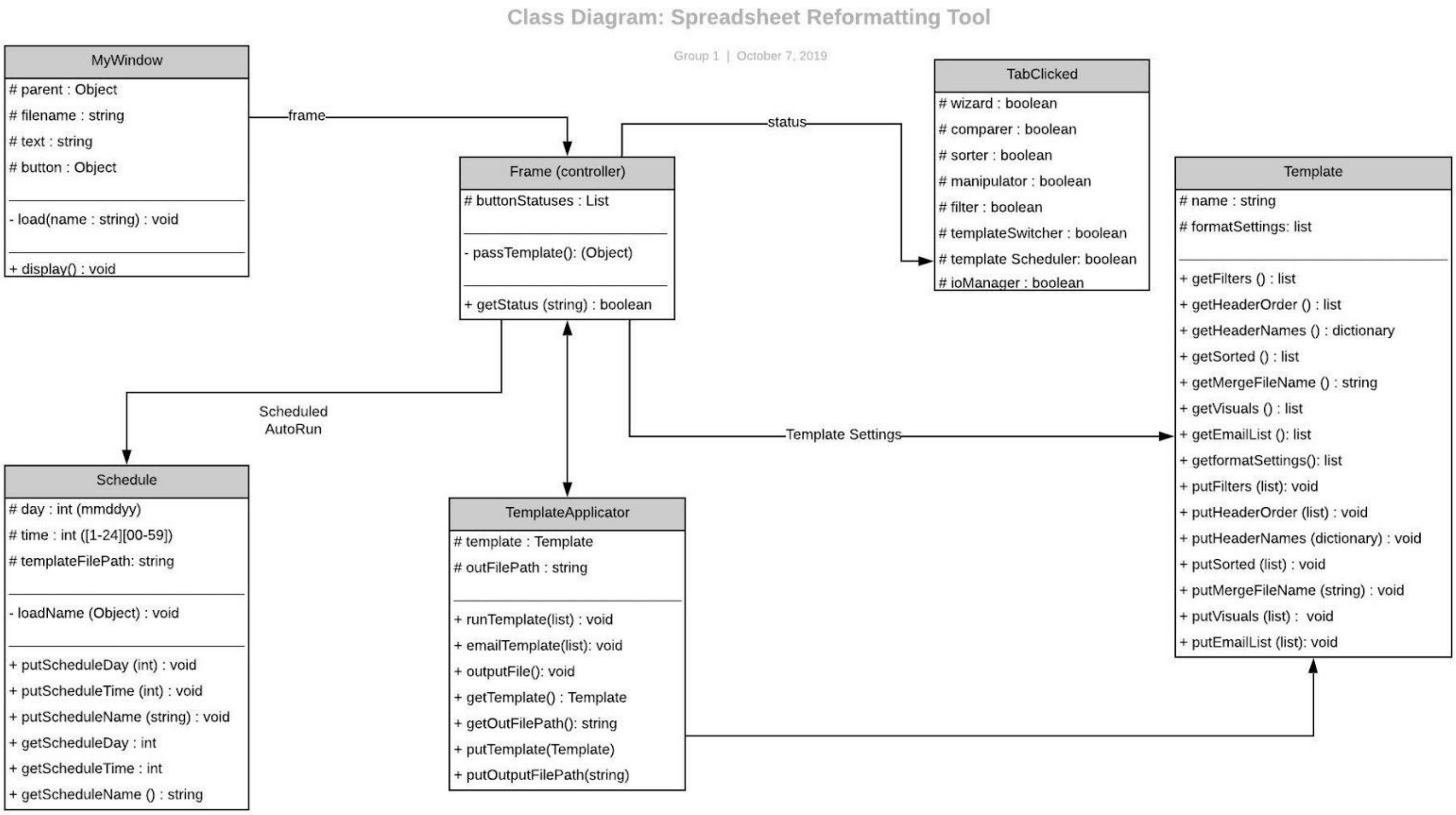


UC-17 Interaction Diagram



2. Class Diagrams, System Architecture and Design

a. Class Diagram



Class Diagram: Spreadsheet Reformatting Tool
Group 1 | October 7, 2019
MyWindow
# parent : Object
# filename : string
# text : string
# button : Object
- load(name : string) : void
+ display() : void
frame
Frame (controller)
# buttonStatuses : List
- passTemplate(): (Object)
+ getStatus (string) : boolean
status
TabClicked
# wizard : boolean
# comparer : boolean
# sorter : boolean
# manipulator : boolean
# filter : boolean
# templateSwitcher : boolean
# template Scheduler: boolean
# ioManager : boolean
Template
# name : string
# formatSettings: list
+ getFilters () : list
+ getHeaderOrder () : list
+ getHeaderNames () : dictionary
+ getSorted () : list
+ getMergeFileName () : string
+ getVisuals () : list
+ getEmailList (): list
+ getformatSettings(): list
+ putFilters (list): void
+ putHeaderOrder (list) : void
+ putHeaderNames (dictionary) : void
+ putSorted (list) : void
+ putMergeFileName (string) : void
+ putVisuals (list) : void
+ putEmailList (list): void
Scheduled AutoRun
Template Settings
Schedule
# day : int (mmddyy)
# time : int ([1-24][00-59])
# templateFilePath: string
- loadName (Object) : void
+ putScheduleDay (int) : void
+ putScheduleTime (int) : void
+ putScheduleName (string) : void
+ getScheduleDay : int
+ getScheduleTime : int
+ getScheduleName () : string
TemplateApplicator
# template : Template
# outFilePath : string
+ runTemplate(list) : void
+ emailTemplate(list): void
+ outputFile(): void
+ getTemplate() : Template
+ getOutFilePath(): string
+ putTemplate(Template)
+ putOutputFilePath(string)

b. Data Types and Operation Signatures

i. Window Class

The window class accesses the buttons, file, and text views and is responsible for window views. Essentially the window is the view to the user.

- parent: Object
  - -The parent object such as a previous window.
- filename: string
  - -The file names used for pulling files.
- text: string
  - -Text that is viewable in the window.
- button: Object
  - -Buttons used for different actions in the window

- load(name : string): void
  -Loads a file using the name of the desired template.
- display(): void
  -The window the user sees.

ii. **Frame Class**

The frame class acts as the controller to the system. It accesses the status of buttons and pulls the status to determine what to do next.

- buttonStatuses: List
  -Tells the statuses of buttons so the program knows what to do next.
- getStatus(string): boolean
  -Retrieves the current status of a button.
- passTemplate(): Object
  -givesTemplate object

iii. Template Class

The template class accesses the name of the templates stored in the system. The template then gets the specified template with the filtered, modified, and sorted data and sets it as the active template.

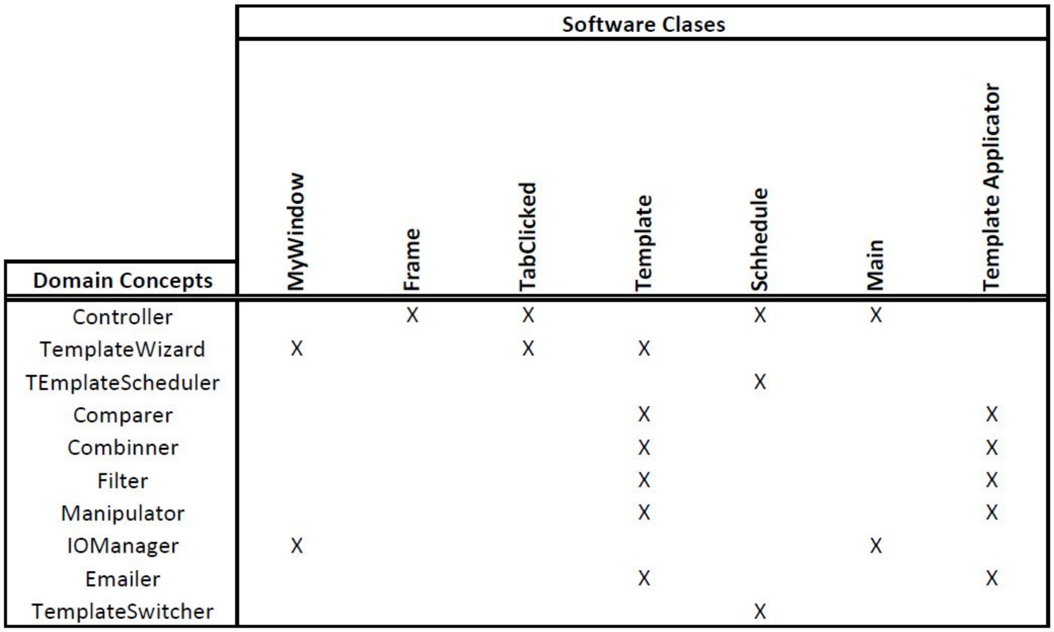
- name: string
  -The name of the templates in storage.
- getFiltered(List): void
  -Gets the current filtered data from the template.
- getModified(List): void
  -Gets any modified data from a template.
- getSorted(List): void
  -Gets the current sorted status from a template.
- setTemplate(): Dictionary
  -Sets the current template being used.

iv. Schedule Class

The schedule class accesses the date and time and pulls a template that is specified at the set date and time.

- date: int(mmddyy)
  -This is the current date in the format of mmddyy.
- time: int([1-24][00-59])
  -This is the current time with a 24 hour clock.
- loadName(Object): void
  -The scheduled object to be run based on the date and time

c. Traceability Matrix



	Software Clases						
Domain Concepts	MyWindow	Frame	TabClicked	Template	Schhedule	Main	Template Applicator
Controller		X	X		X	X	
TemplateWizard	X		X	X			
TEmplateScheduler					X		
Comparer				X			X
Combinner				X			X
Filter				X			X
Manipulator				X			X
IOManager	X					X	
Emailer				X			X
TemplateSwitcher					X		

3. System Architecture and Design

a. Architectural Styles

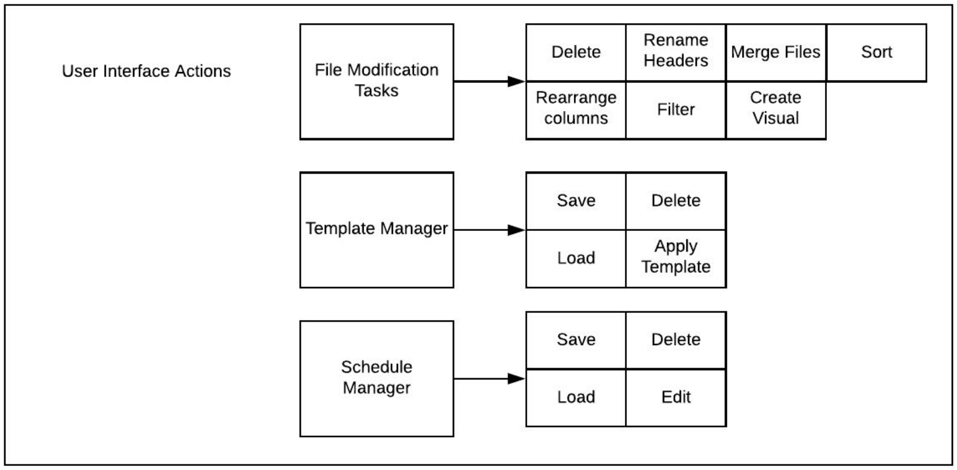
Our system is designed with the architectural style called “Component-based software engineering”. The program is mainly a collection of different functions that are to be applied to a report object (spreadsheet/table). As such, the different functions are implemented in isolation, as components. They all work to modify the given report object, upon the user’s discretion. Information from each modification task is stored temporarily in the program, and then later compiled with other modifications (i.e. deleted columns and the filters applied) and stored into a template for later use. Upon use of the template (or a scheduled run of a template), these individual parts are dispersed to the appropriate modules for altering the file.

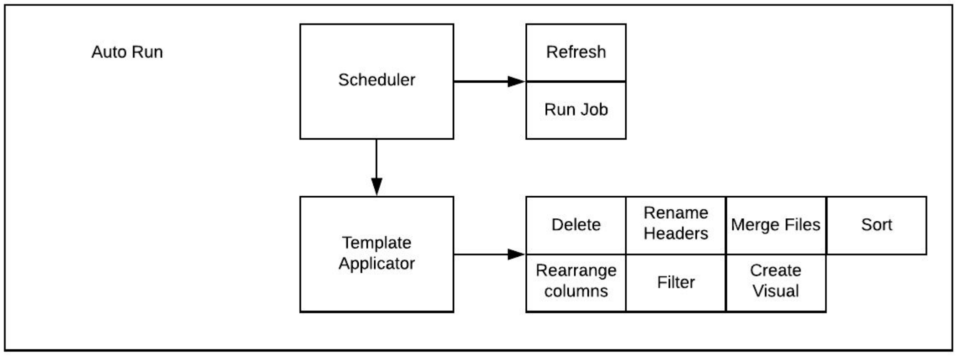
b. Identifying Subsystems

Within our system, there are two routes for the main application: user-directed actions via the interface, and automated runs from the scheduler or taken from a user-directed template application (user selects a stored template and that template is applied to the file it references, rather than the template application being executed from the result of a schedule).

The file modification tasks subsytem will make available to the user all the possible modifications to a file, to be performed one at a time by the user. The template manager will allow the user to save the current settings as a template, load an existing, delete an existing, or apply a template. When the user chooses to apply a template, the template manager passes control to the Auto-Run subsystem.

The auto-run subsystem handles template applicables generated as the result of the user selection or from a scheduled job. In cases of user selection, the scheduler does not come into play, control is passed directly to the template applicator, and all changes made. The scheduler is responsible for monitoring the pending jobs and passing the applicable template to the template applicator when needed.





c. Persistent Data Storage

Template and schedule objects will need to be saved outside of the system for later access. Template objects will be stored in a flat file format. For easy storage and retrieval, the template files will be serialized and stored in the ".pickle" format. Schedules will also be stored for later access in the pickle format. The main program will load the schedule file, and subsequently, the template files will be opened as applicable.

d. Global Flow Control

Overall, the system can be seen as a linear flow, in viewing the procedure as "load file", "edit file", "save template". However, within the "edit file" step, there is an event-driven flow. Although the user is guided through the same sets of steps in setting up a file format, each user may perform the same task type multiple times (i.e. deleting multiple fields rather than just one). Additionally, users may freely navigate from one part of the guided editor to another, so there is no particular order for most of the file formatting. The system waits to respond to various user input.

Additionally, there are time dependencies within our system, as it will incorporate scheduling of events (without interaction for the user). The scheduler will activate the program and supply the template to the program for processing. The scheduler will routinely check for pending jobs and execute when a job is awaiting processing.

e. Hardware Requirements

The program requires the following of the end-user's computer at a minimum. Larger files will require more working memory and the program may run slowly if only the minimum requirements are used.

- i. CPU: 1GHz or faster
- ii. Memory: 2GB
- iii. Hard Disk: 100MB for installation
    1. Additional hard disk space if storing output files on personal computer and not network location
- iv. Display: Any display supported by the individual computer