System Programming Project - Deliverable 1

EasySave version 1.0

31/01/2022

CESI Engineering School

Saint-Nazaire

Anne-Laure GAUDON : Pedagogical tutor

FISA A3 Computer Science - Group 2

COUTEAUX Corentin

DEMBELE Romaric

RAMBIER Ewen

Method of dissemination:
Free

TABLE OF CONTENTS

INTRODUCTION

The advent of digital technology in our societies has revolutionized the way of working. Data is increasingly backed up thanks to software that offers this functionality. It is in this context that EasySave, a software for backing up data that is in the form of computer files, is placed. The realization of this application requires prior software modeling, a number of tools and computer technologies. Also, a user manual of the software must allow an easier and more efficient use of the latter.

I. UML DIAGRAMS

To get a visual overview of software and its features from the perspective of a developer and a non-developer, a number of standardized diagrams exist. The Unified Modeling Language (UML) is used to create these diagrams. For our software, a presentation of the 4 of these diagrams will be made.

I.1 Use case diagram

Figure 1: Use Case Diagram

The use case diagram provides an overview of the different features of the software. It is a diagram that can be presented to project actors who do not necessarily have a technical background.

Thus, we can see in the image above the main features of the software namely: running a backup job, sequentially running all backup jobs, creating, renaming and deleting a backup job as well as changing the language of the software.

I.2 Activity diagram

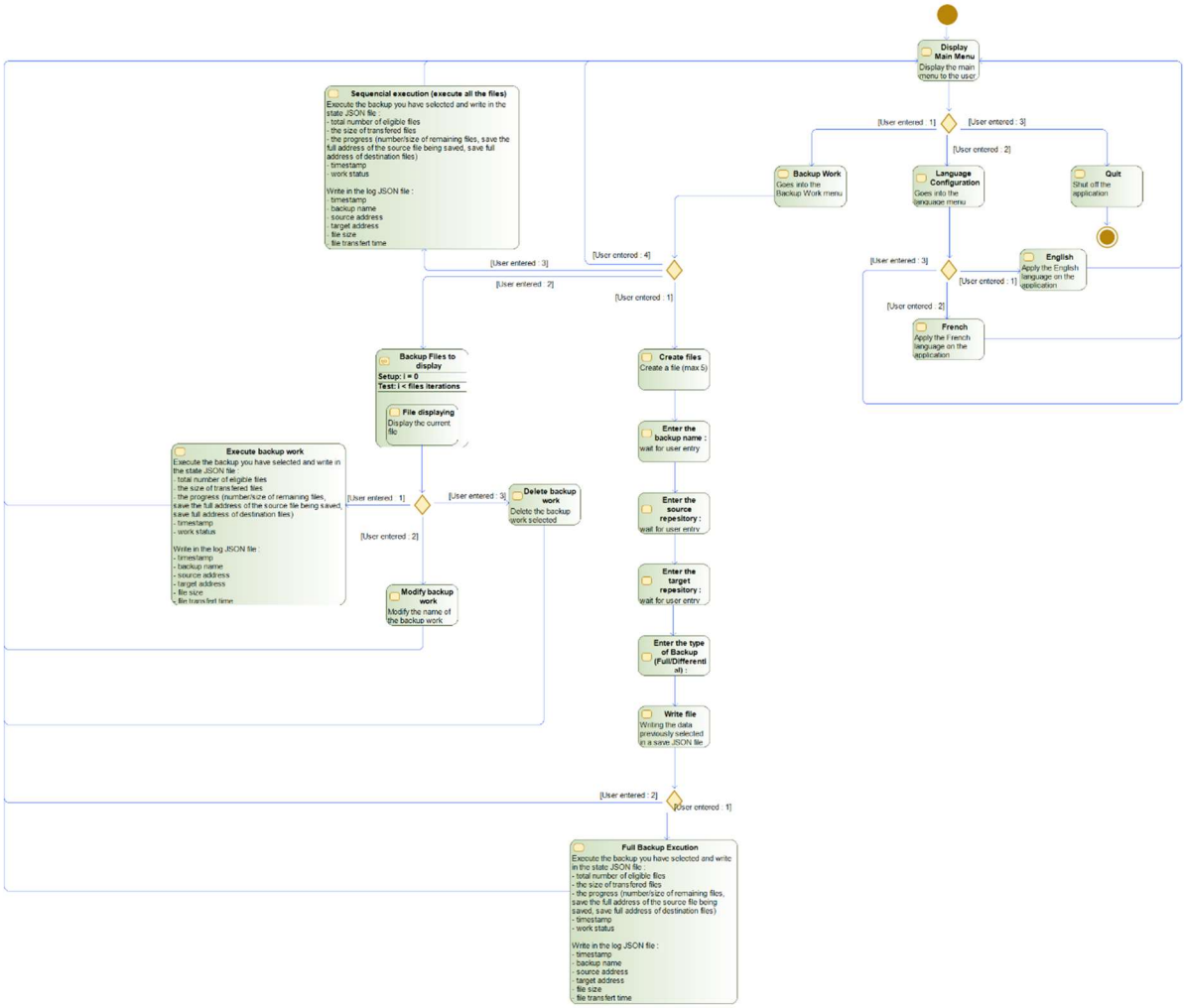

Figure 2: Activity Diagram

An activity diagram is a diagram showing the triggering of events based on system states. It allows to express a temporal dimension of the use cases of the software. Representing a software use case by an activity diagram is like algorithmically translating the use case.

Thus, we can from the diagram above (Figure 2) see that from the main menu of the application, the user has the choice between leaving the application (the user types the number 3 on his keyboard), configuring the language of the software (the user types the number 2 on his keyboard), and performing a backup job (the user types the number 1 on his keyboard) . Depending on his choice, several other choices are presented to him (refer to the diagram to see all the possibilities).

I.3 Class diagram

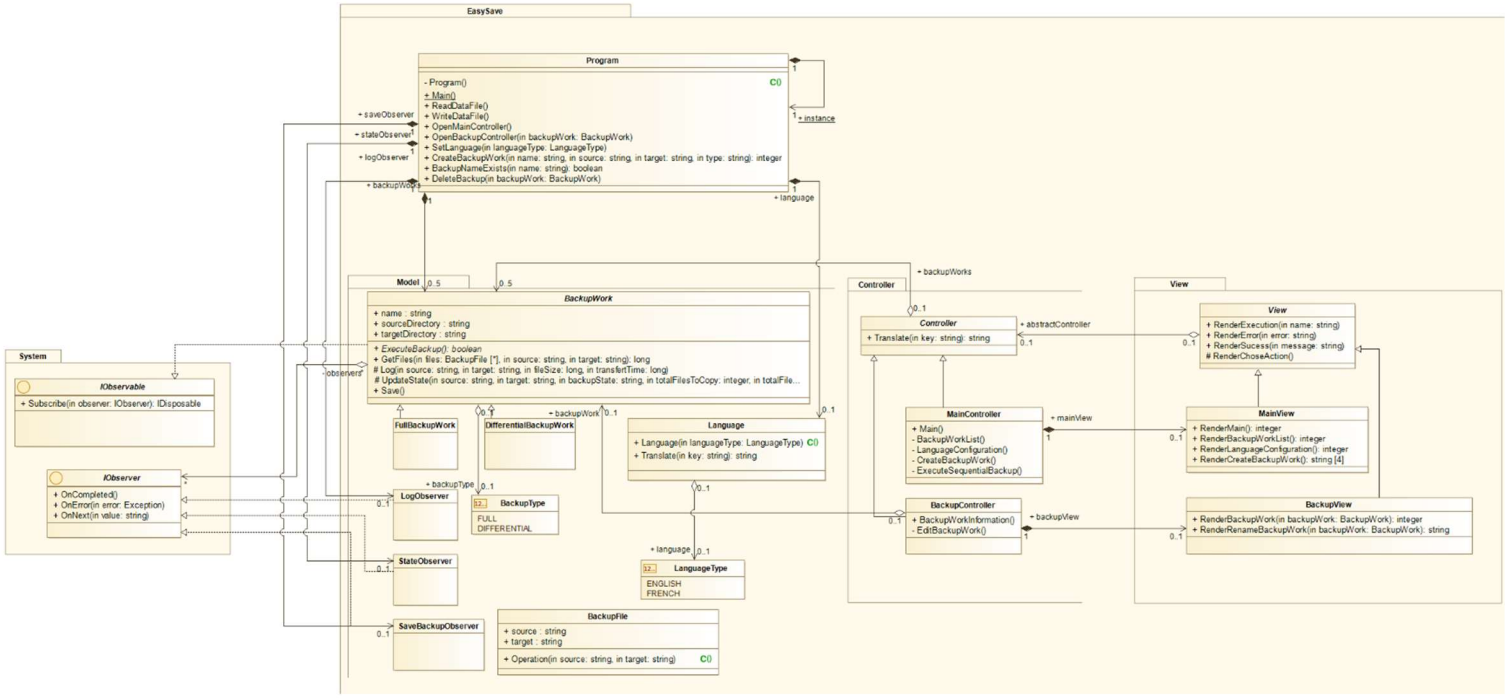

Figure 3; Class diagram

In order to have the structure of the software to be implemented, the class diagram intervenes by showing the classes and interfaces of the program as well as the relationships between the different classes that can be of type association, aggregation, composition and inheritance.

The class diagram of our software presents the classes by grouping them according to their nature in the MVC (Model View Controller) architecture. The architecture of the software is presented in five main parts namely the Model, View, Controller, Program and System part.

I.4 Sequence diagrams

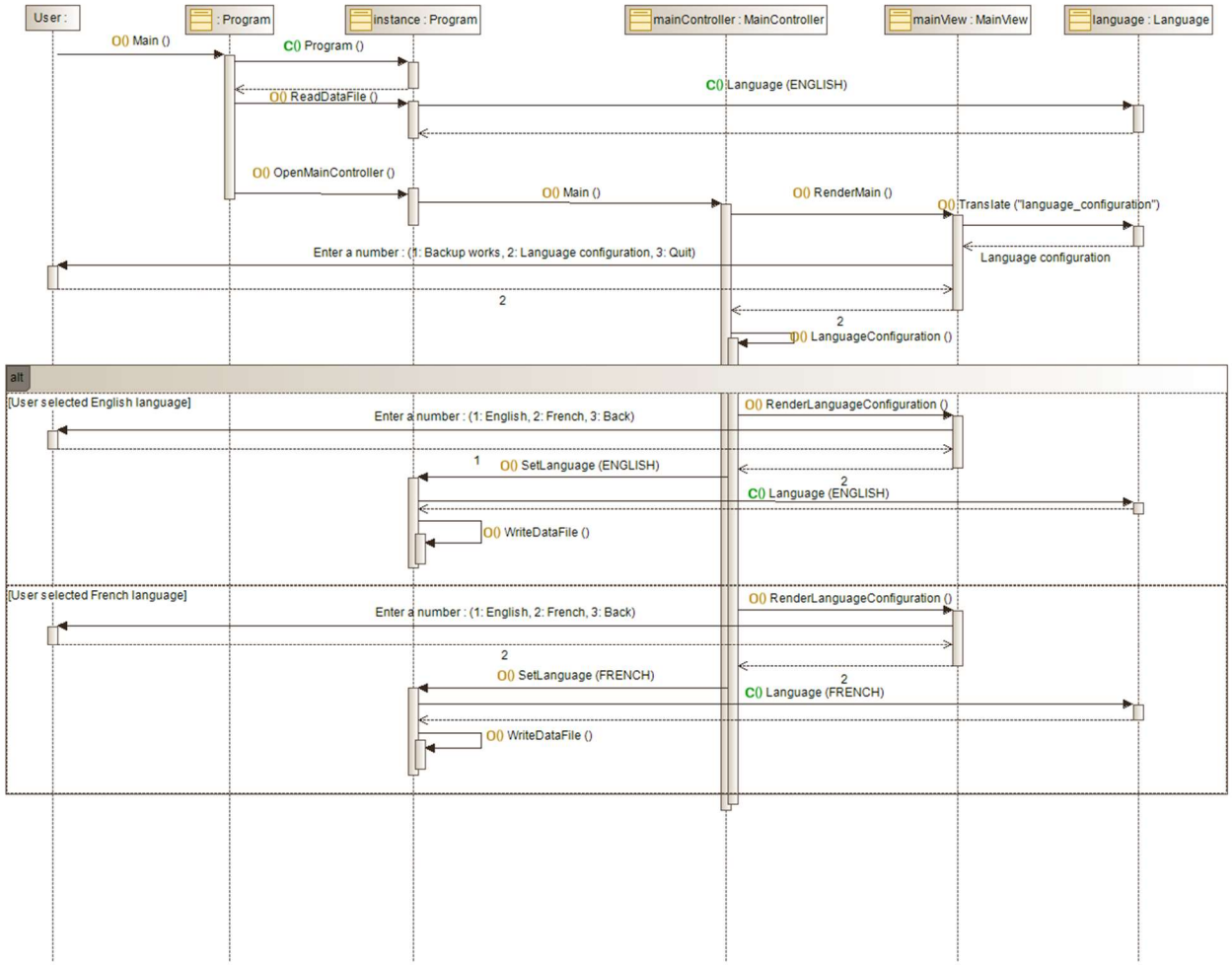

Figure 4: Sequence diagram – language choice

The sequence diagram is a diagram illustrating the chronological interactions between objects in a system (in our case backup software) for a given software use case.

I.4.1 Choice of language

Figure 4 illustrates, from the sequence diagram, the necessary interactions between the objects of the software in order to make a language change (two possible language options – English and French).

I.4.2 Sequential execution

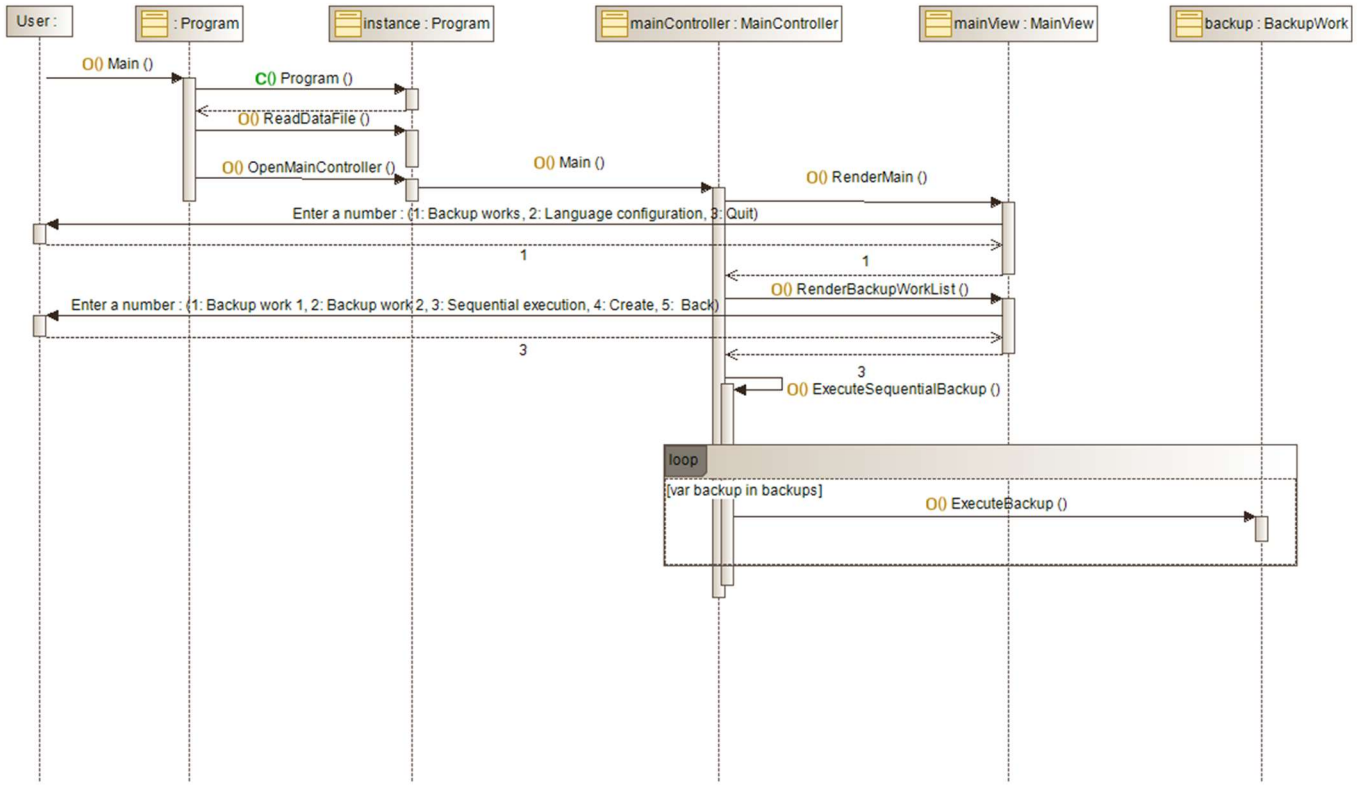

Figure 5: Sequence Diagram – Sequential Execution

Figure 5 illustrates the interactions required between the user and the software objects in order to perform a sequential backup job.

I.5 Component diagram

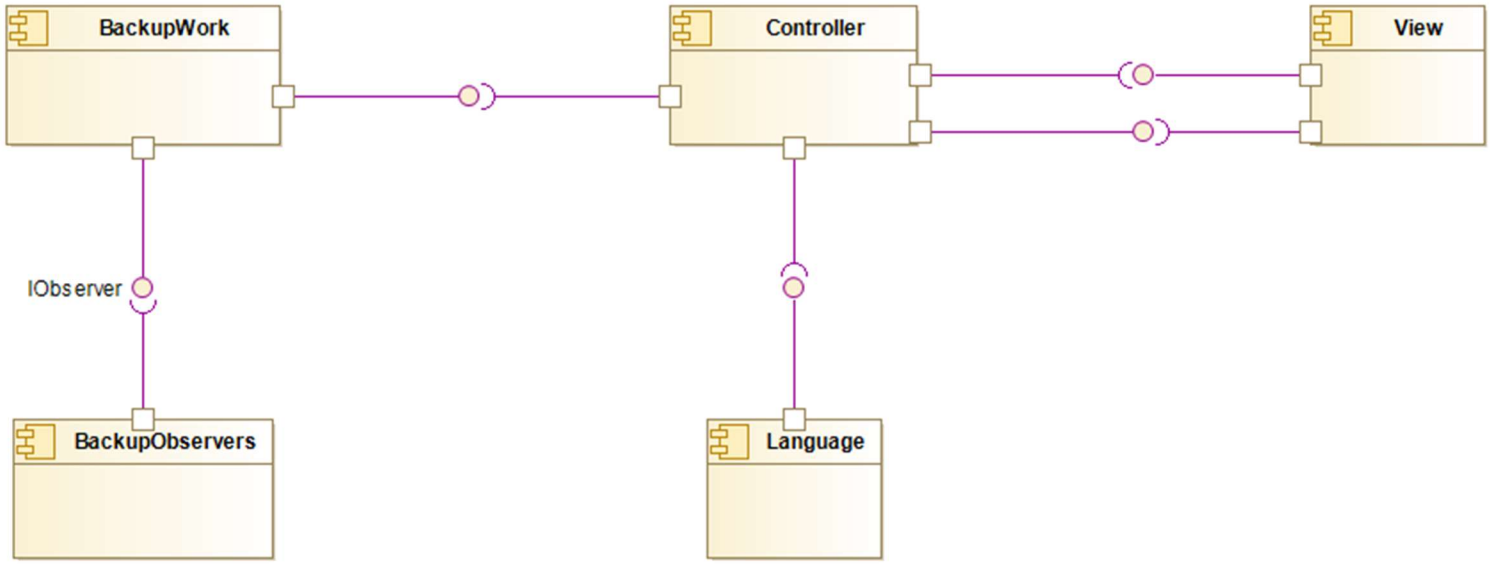

Figure 6: Component Diagram

The component diagram is a diagram showing an overview of the system components and the dependencies between those components. Components can be classes, class sets, packages, software libraries, and so on.

Our diagram (Figure 6) shows the components (BackupWork, Controller, View BackupObservers, Language) and their mutual dependencies.

II. TOOLS AND TECHNOLOGIES USED

II.1 Design patterns

In order to facilitate software design, design patterns that have already proven themselves to software developers, come into play. There are a multitude of design patterns; each solving a specific problem. According to Wikipedia, a design patron is a caracteristic arrangement of modules, recognized as good practice in response to a software design problem.

Regarding version 1 of our backup software, two design patterns namely the **MVC** architecture (Model - View - Controller) and the **"Observer"** pattern have been implemented.

The MVC architecture is an architecture motif segmented into three layers namely:

- A template containing the data to display.
- A view containing the presentation of the software interface.
- And a controller corresponds to the logic of the user's actions.

The "Observer" design pattern is used to send signals to software modules called observers to take actions based on these signals (notifications). In the case of our backup software, it is used for updating logs.

II.2 DevOps

II.2.1 Azur DevOps GIT

In order to have a versionnage of our backup software during the design as well as to allow collaborative work we used GIT which is a decentralized version management software.

II.2.2 GitHub

The remote development management of our software was carried out with GitHub.

II.3 Development Platform

The development of the first version of our software was carried out on Visual Studio 2019 with the .Net Core 3.X Framework

II.4 NuGet Package Newtonsoft. Json

In order to convert a .Net object to a Json object, we use the **Newtonsoft.Json** NuGet package in our software.

III. USER DOCUMENTATION

The software provides the following features to a user:

```
1. Backup works
2. Language configuration
3. Quit
Choose an action : 2
1. English
2. Français
3. Back
Choose an action : 2
Langue définit en Français
1. Travaux de sauvegarde
2. Configuration de la langue
3. Quitter
```

Figure 7: Language Change

Change of language

When starting the software, the user has the possibility to change the language by tapping on the number 2, then on the Enter key on his keyboard and then choose his language (Figure 7).

```
C:\workplace\ProgrammationSysteme\bin\Debug...
1. Backup works
2. Language configuration
3. Quit
Choose an action : 1
1. Create a backup work
2. Back
Choose an action : 1
Enter the backup work name : Save1
Enter the source repository : C:\Users\DEMBELE Romari
c\Desktop\CESI 2021-2022\COURS\Programmation_Systeme\
Projet\SourceForlder
Enter the target repository : C:\Users\DEMBELE Romari
c\Desktop\CESI 2021-2022\COURS\Programmation_Systeme\
Projet\DestinationFolder
Enter the backup type (full / differential) : full
Backup work created
1. Save1
2. Execute all backup works
3. Create a backup work
4. Back
Choose an action : 1
Backup work selected : Save1
Source : C:\Users\DEMBELE Romaric\Desktop\CESI 2021-2
022\COURS\Programmation_Systeme\Projet\SourceForlder
Target : C:\Users\DEMBELE Romaric\Desktop\CESI 2021-2
022\COURS\Programmation_Systeme\Projet\DestinationFol
der
Type : Full
1. Execute the backup
2. Edit
3. Delete
4. Back
Choose an action : 1
Execution of Save1
Backup completed
Backup work selected : Save1
Source : C:\Users\DEMBELE Romaric\Desktop\CESI 2021-2
022\COURS\Programmation_Systeme\Projet\SourceForlder
Target : C:\Users\DEMBELE Romaric\Desktop\CESI 2021-2
022\COURS\Programmation_Systeme\Projet\DestinationFol
der
Type : Full
1. Execute the backup
2. Edit
3. Delete
4. Back
Choose an action :
```

Figure 8: Creating and Running a Job backup

Creating, Executing, Deleting a Backup Job

After creating a backup job (during creation, the type of backup can be chosen – Full or differential), the user can decide to run, edit or delete it by following the different options offered on the screen (See Figure 8 opposite)."

Performing all backup jobs

The user will also be able to decide to run all the backup jobs he has previously created if he wishes.

(The option arises if he has already created at least one save job).

CONCLUSION

By way of conclusion, through the diagrams of use cases, activity, classes, sequence and components, the modeling of the EasySave backup software allowed an easier implementation, taking into account all the specifications of the specifications. When it comes to software versioning and collaborative work, the GIT and GitHub tools have been a great asset.

The segmentation of the software code allowed to benefit from a modularity and scalability of the software. Moreover, the second version of the software, which this time will have a graphical interface, will rely on this modularity for a faster and easier implementation.