HUBLAU Alexandre
DA SILVA CARMO Alexandre
PALMIERI Adrien
CHEVRIER Jean-Christophe

PROJET TUTORÉ

-

DOCUMENTATION

-

PREMIÈRE APPROCHE DU SUJET

SOMMAIRE

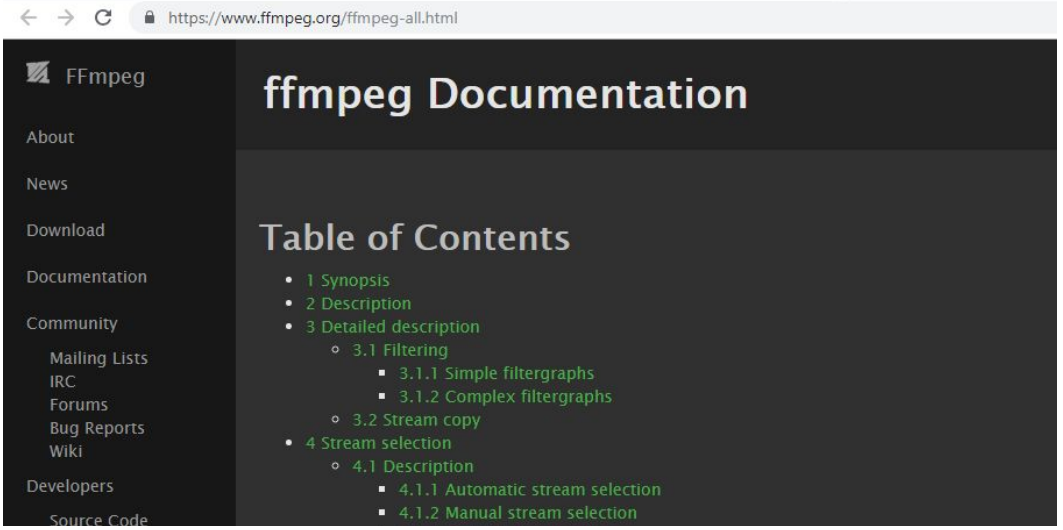
1) Description détaillée du sujet

Le sujet de notre projet est de **réaliser une interface graphique pour le logiciel FFmpeg**.

FFmpeg est une collection de logiciels libres, permettant de réaliser des traitements sur des vidéos. Le problème de ce logiciel est qu'il n'est pas adressé à l'utilisateur lambda. En effet ce logiciel ne possède pas d'interface graphique, tout traitement sur une vidéo doit se faire par l'intermédiaire de lignes de commandes. Ce qui n'est pas du tout intuitif pour un utilisateur peu coutumier de l'invite de commandes sous Windows, ou même du terminal sous Linux. Ainsi, il est dommage d'observer qu'un tel logiciel, en open source, ne puisse pas profiter à tout le monde pour cette unique raison. Il y a donc clairement un problème qui se pose : **Rendre FFmpeg accessible à tous utilisateurs**.

FFmpeg est un logiciel proposant de nombreuses fonctionnalités, comme couper une vidéo en plusieurs sous séquences de vidéos. Il permet également de rogner des vidéos, d'en extraire le son, d'ajouter un son, de convertir en un nouveau format. L'ensemble des fonctionnalités proposées par FFmpeg, que nous avons testés pour le moment, **sera listé un peu plus tard dans le document** (cf : 3.3) Liste des fonctionnalités testées).

Il faut noter que la documentation portant sur FFmpeg, proposée par ses développeurs, disponible sur le site officiel de celui-ci, est en **anglais**, **et est particulièrement conséquente**. Il faut donc prendre en compte que nous n'avons probablement pas survolé la totalité des possibilités qu'offre ce logiciel. Nous nous sommes pour le moment concentrés sur ses principales fonctionnalités. Par exemple à partir de ce lien : https://www.ffmpeg.org/ffmpeg-all.html on peut consulter la documentation consacré uniquement aux fonctionnalités de FFmpeg sans ces extensions (ffplay, etc.). C'est complet, pour ne pas dire "**excessivement complet**".



Nous avons donc pour mission dans ce projet de réaliser une interface graphique pour ce logiciel de montage vidéo. Cette interface doit être ergonomique mais en même temps intuitive. Un objectif serait d'**implémenter un maximum de fonctionnalités dans l'interface tout en conservant une interface pas trop surchargée et simple d'utilisation**. Il faut chercher le meilleur rapport "**fonctionnalités/interface**".

Une question qui se pose est : "**Allons-nous nous contenter d'adapter le logiciel, ou allons-nous voir loin, et imaginer de nouvelles fonctionnalités ?**". Cette question reste en suspens pour l'instant, elle trouvera probablement une réponse dans les itérations futures.

2) Étude des solutions informatiques déjà existantes

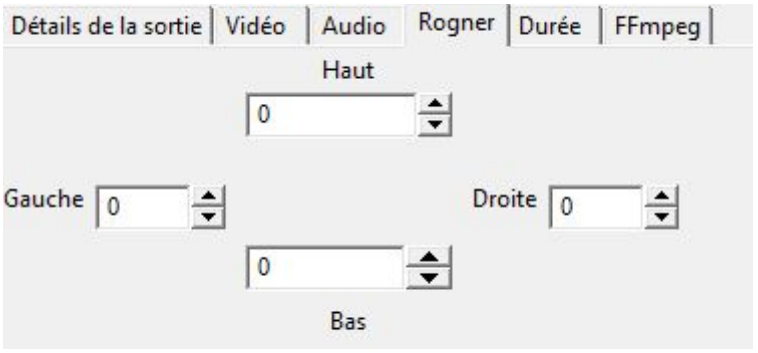
Nous avons pu observer qu'il existe déjà des solutions répondant au problème expliqué plus tôt. En effet des logiciels déjà existant propose des interfaces graphiques pour FFmpeg. Nous avons donc réalisé une "étude de l'existant", ou plutôt des "existants". Il faut noter que chacune de ces solutions a ses avantages et ses inconvénients.

2.1) WinFF, une solution sous linux et windows

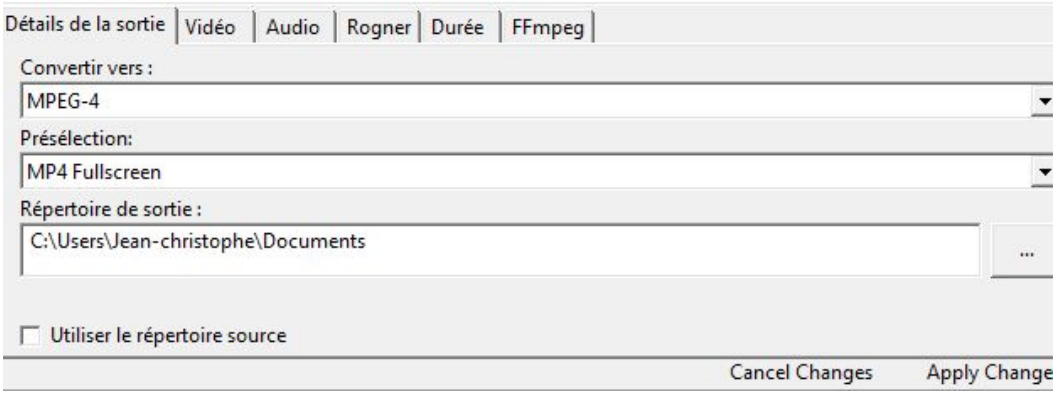
2.1.1) Présentation de WinFF

WinFF est une interface graphique écrite en Free Pascal pour utiliser le logiciel ffmpeg sous Windows et Linux. **Notre avis est mitigé, cette interface peut être pratique pour certaines fonctionnalités et pas du tout pour d'autres**. Par exemple, convertir une vidéo s'avèrera très simple.

En revanche, **rogner une vidéo est un "calvaire"**, on devrait pouvoir rogner directement sur la vidéo, et pas à avoir à rogner depuis cette interface :



De plus on peut noter, que esthétiquement parlant l'interface que nous propose WinFF est “**primaire**”.



2.1.2) Analyse de l’interface

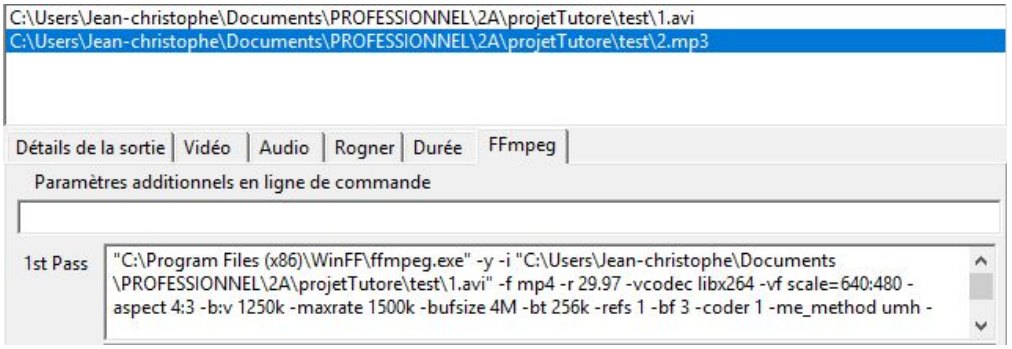
Les boutons permettent d’ajouter des médias, de retirer un média importé, de retirer tous les médias importés, de lire un média importé, d’afficher un aperçu de la vidéo/son final, et de convertir une vidéo. On appelle média : son, vidéo, image , ou même sous-titre.

L’onglet “**Détails de sortie**” permet de choisir les formats de vidéos en entrée et en sortie ainsi que choisir le répertoire et le nom du fichier de destination.

L’interface possède différentes options : **“Vidéo”, “Audio”, “Rogner”, “Durée”, “FFmpeg**”.

Les options ”Vidéo” et “Audio” sont assez explicites. L’option “Rogner“ permet de rogner une vidéo dans tous les sens comme ca a été montré précédemment (cf : page 4). L’option “Durée” permet de couper une vidéo.

Enfin il est possible avec l’option “FFmpeg” **d’accéder à la commande ffmpeg que nous sommes en train de réaliser avec WinFF, et d’ajouter des paramètres à la commande.** Cela est utile dans le cas où le paramètre que l’on veut ajouter n’est pas ajoutable par l’intermédiaire des outils que propose WinFF.



2.2) TraGtor, une solution sous Linux

2.2.1) Présentation de TraGtor

TraGtor est une interface graphique écrite en langage Python pour utiliser le logiciel FFmpeg sous linux. Il s'agit donc d'une solution utilisable uniquement sous des distributions linux, **elle n'est pas multiplateforme.**.

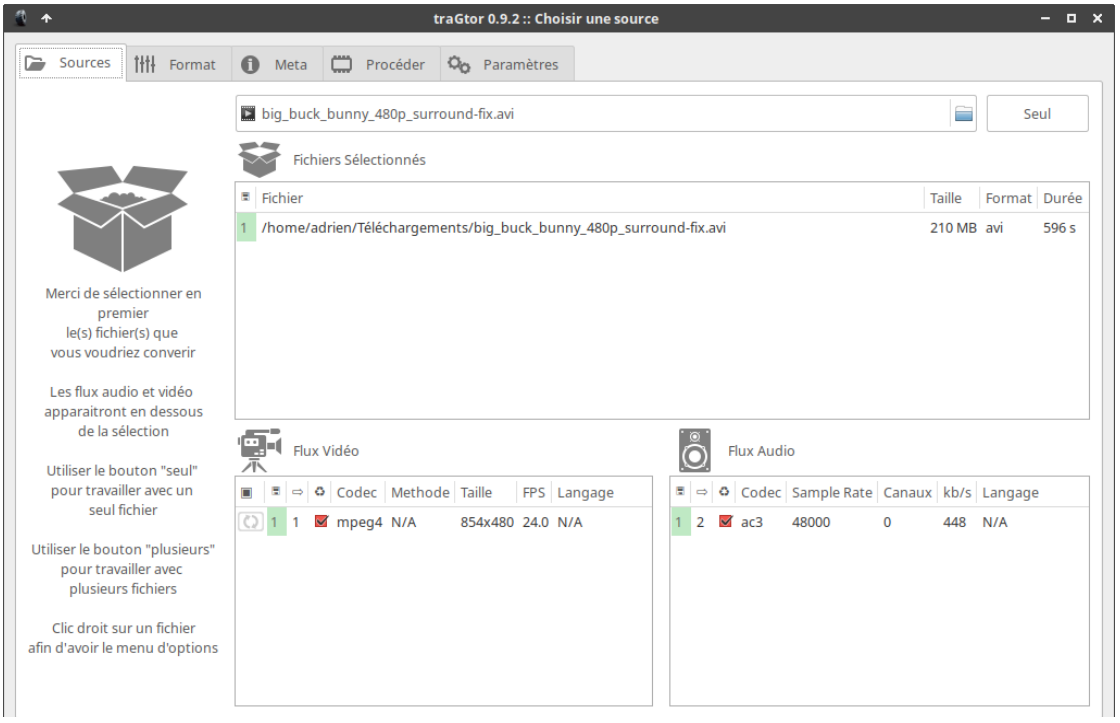
Son objectif **n'est pas d'implémenter toutes les fonctionnalités de FFmpeg** mais de permettre à l'utilisateur de convertir facilement un fichier multimédia dans un autre format (vidéo -> vidéo / vidéo -> audio / …). Il ne faut donc pas chercher à pouvoir utiliser toutes les fonctionnalités de FFmpeg avec ce logiciel.

Cependant, TraGtor répond à ce problème en permettant d'effectuer des opérations plus poussées si l'utilisateur connaît les commandes de ffmpeg, puisqu'**il est possible de modifier la commande générée par l'interface graphique avant le démarrage de la conversion**. Mais cela reste du coup peu intuitif pour l'utilisateur final, qui doit consulter la documentation de FFMpeg pour effectuer des opérations plus complexes.

2.2.2) Analyse de l'interface

L'interface du logiciel est divisée en 5 onglets : **Sources, Format, Meta, Procéder, Paramètres**. Le premier "Sources" permet de choisir le ou les fichiers sources, ainsi que les différents flux des sources sur lesquelles nous souhaitons travailler (audio/vidéo). Le second onglet "Format" quant à lui permet de configurer les différents paramètres pour le format de sortie : codec utilisé, résolution de la vidéo, bitrate audio, partie de la vidéo sur laquelle nous travaillons, utilisation d'options avancées. Le troisième "Meta" permet d'ajouter des métadonnées sur le fichier (titre, auteur, copyright). Le quatrième onglet "Procéder" permet de choisir tous les paramètres de sortie tels que le nom du fichier ainsi que son emplacement, pour finalement lancer l'opération. Le dernier onglet "Paramètres" permet de modifier les paramètres généraux du programme (langue, gestion de la compatibilité avec ffmpeg, options de l'interface).

Cette interface est **simpliste** : elle guide l'utilisateur étape par étape dans son projet de conversion. On peut également ajouter que d'un point de vue esthétique, cette interface est bien plus agréable que celle de WinFF, qui est peu soignée.



2.3) Synthèse des problèmes des solutions existantes

Au final, on peut en conclure que ces deux interfaces ont chacune leur problèmes. WinFF et son aspect graphique peu soigné, traGtor qui n'implémente pas certaines fonctionnalités pourtant principales comme le fait de pouvoir rogner une vidéo, ou encore d'extraire simplement l'image d'une vidéo. Ces deux interfaces mériteraient d'être **plus approfondies**.

Par exemple, Il faudrait pouvoir rogner, ou même couper une vidéo directement sur la vidéo. L'utilisateur veut voir concrètement en image à quoi correspond le moment où il coupe la vidéo, et à quel moment correspond le moment dans la vidéo où il arrête de couper. Sans parler de l'action rogner, **comment rogner pourrait être intuitif si on ne voit pas ce que l'on rogne**. **Pour n'importe quel utilisateur lambda cela semblerait**

déboussolant, et même aberrant. Et pourtant c'est ce que propose ces deux logiciels, **il faudra donc que nous évitions de reproduire ces "erreurs" dans notre projet**.

3) Étude technique de FFmpeg

3.1) Recherche et étude du logiciel

Afin de gagner des connaissances sur notre sujet, nous avons réalisé une première étude du logiciel : nous avons consulté des sites et des articles contenant des renseignements sur FFmpeg, ainsi que le site officiel de celui-ci qui fournit une documentation gargantuesque sur le logiciel et sur ses extensions (https://www.ffmpeg.org/) . Nous avons donc pu retenir un ensemble de fonctionnalités intéressantes, la manière de les utiliser en ligne de commandes, et ainsi de les tester.

3.1.1) Tests

Dans ces fameuses lignes de commandes, **il faut toujours préciser le fichier source avec l'option "-i" ainsi que le fichier de destination**. Voici un exemple : `C:\Users\Jean-christophe>ffmpeg -i input.mp4 output.avi`, il s'agit d'un exemple de conversion de fichier .mp4 en .avi.

Il est possible, selon les traitements vidéo que vous souhaitez réaliser, que vous ayez besoin de préciser quelles parties de chaque vidéo source vous voulez conserver dans le fichier de destination. Le son, l'image, et les sous-titres forment par exemple différentes parties d'une vidéo, FFmpeg dans sa documentation appelle ces parties des "stream", c'est-à-dire : des **flux**. FFmpeg propose une option pour préciser les flux que l'on souhaite conserver : **-map indiceVideoSource : indiceFlux**.

Les indices des vidéos et des flux commencent à 0. Voici un exemple :

```
>ffmpeg -i video.mp4 -i audio.mp3 -c copy -map 0:0 -map 1:0 video_finale.mp4
```

Dans ce cas on prend l'image de vidéo.mp4, et le son de audio.mp3 pour faire video_finale.mp4, le son de video.mp4 n'a pas été conservé, pourtant on aurait très bien pu fusionner le son de video.mp4 et de son.mp3 pour en faire le son de video_finale.mp4.

Ceci était une légère introduction à toutes les capacités de FFmpeg, nous n'allons évidemment pas tout expliquer en détail, puisque ceci a déjà été très bien fait par les développeurs de celui-ci, dans la documentation de leur page officielle.

Ainsi nous avons chacun de notre côté testé des fonctionnalités. Le site : https://www.linuxtricks.fr/wiki/ffmpeg-la-boite-a-outils-multimedia#paragraph_extraire-l-audio-

d-une-videohttps://trac.ffmpeg.org/ est un très bon site pour cerner les principales fonctionnalités de FFmpeg et pour débuter avec.

3.1.2) Solutions envisageables

Pour répondre à la problématique posée par notre sujet, deux solutions semblent s'offrir à nous. La première est programmer en JAVA l'interface graphique. La deuxième solution est de programmer l'interface en C++, solution possible puisque le logiciel FFmpeg a été écrit en C: il est possible d'intégrer des librairies C dans des projets en C++.

Nous aurions pu également effectuer une interface graphique directement en C avec GTK par exemple**, mais la gestion des threads et l'absence d'objets aurait rendu le processus bien plus complexe**. De plus les développeurs de FFmpeg, sur leur site officiel, conseille d'utiliser C++ pour implémenter de nouvelles fonctionnalités dans leur logiciel.

La première solution étant de programmer en JAVA, elle incombe de développer une interface indépendante du code source de FFmpeg. Mais l'avantage est que JAVA est un environnement que nous connaissons et avec lequel nous avons déjà développé des **interfaces homme-machine**. Dans ce cas, le lien entre les classes programmant l'interface et les bibliothèques FFmpeg se ferait par des appels système vers FFmpeg dans les classes. Cela peut poser problème, car la **gestion les erreurs retournées par FFmpeg sera plus compliquée** puisque nous n'accederons pas directement aux bibliothèques, mais aux messages d'erreurs retournés par les appels système. De plus le fait de devoir faire tourner du code java et FFmpeg en même temps **risque d'augmenter légèrement le temps d'exécution de chaque tâche.** Nous nous sommes renseignés sur la manière de réaliser des appels système en JAVA, et voici un morceau de code qui fonctionne pour exécuter des commandes FFmpeg directement dans un programme JAVA :

```java
import java.io.*;

public class test {

    public static void main(String[] args) throws IOException {
        String prog = "C:\\Users\\Jean-christophe\\Documents\\PROFESSIONNEL"
                + "\\2A\\projetTutore\\ffmpeg-20181031-4a97620-win64-static\\bin\\ffmpeg ";

        String source = "C:\\Users\\Jean-christophe\\Documents\\PROFESSIONNEL"
                + "\\2A\\projetTutore\\test\\1.mp4 ";

        String destination = "C:\\Users\\Jean-christophe\\Documents\\PROFESSIONNEL"
                + "\\2A\\projetTutore\\test\\1.avi ";

        Runtime rt = Runtime.getRuntime();

        rt.exec(prog+ "-i  "+source+destination);
    }

}
```

(Ceci est un simple exemple de conversion. Il est possible également d'utiliser un objet **Process** pour récupérer les réponses renvoyées par la commande FFmpeg.)

La seconde solution étant de programmer en C++, langage compatible avec les bibliothèques de FFmpeg, nécessite de l'expérience dans celui-ci : des connaissances telles que la compilation des librairies de FFMpeg pour les intégrer dans un projet en C++, ainsi que l'apprentissage de la gestion de la mémoire et d'une IHM en C++ seront nécessaires pour mener à bien le projet de manière correcte. Rappelons que pour l'instant nous n'avons de l'expérience que dans le langage C. Nous nous nous rendons compte que c'est dommage de ne pas pouvoir programmer directement dans le langage initiale du logiciel. **Mais il y a un vrai enjeu, car si nous choisissons de programmer en C++, pour ainsi dire nous ne sommes pas sûrs de vers quoi nous nous dirigeons**. De plus, FFMpeg dispose d'une **documentation spécifique** appelée **Doxygen,** pour les développeurs externes, expliquant les **différentes méthodes** que l'on peut appeler en implémentant FFMpeg et disposant de certaines **programmes d'exemples basiques** sur l'utilisation de celle-ci (attention cependant, ces exemples sont écrits en C et seront donc à réadapter).

Pour l'instant nous sommes favorables à la première solution. Cependant, nous pensons étudier sur le côté le C++. Nous allons nous engager avec la première solution, mais prenons soin de garder à notre chevet la seconde solution (en complétant nos connaissances sur le C++). .

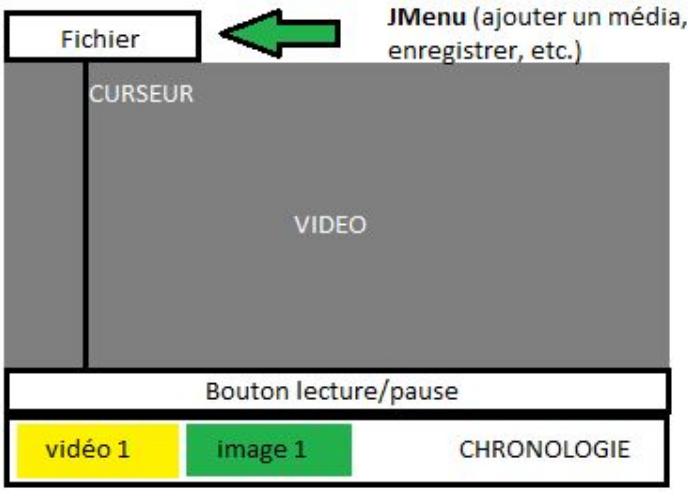
Une bonne idée serait de développer notre interface de manière **flexible**, de telle sorte qu'elle fonctionne avec des opérations en **tâche de fond**. Ainsi l'utilisateur pourrait par exemple lire une vidéo tout en important un nouveau média vers le logiciel en même temps : il y gagnerait en temps. L'objectif serait de **minimiser le temps d'exécution** de chaque tâche et d'avoir un **maximum de fonctionnalités implémentées par l'interface**, le tout en gardant une interface intuitive.

3.2) Première itération

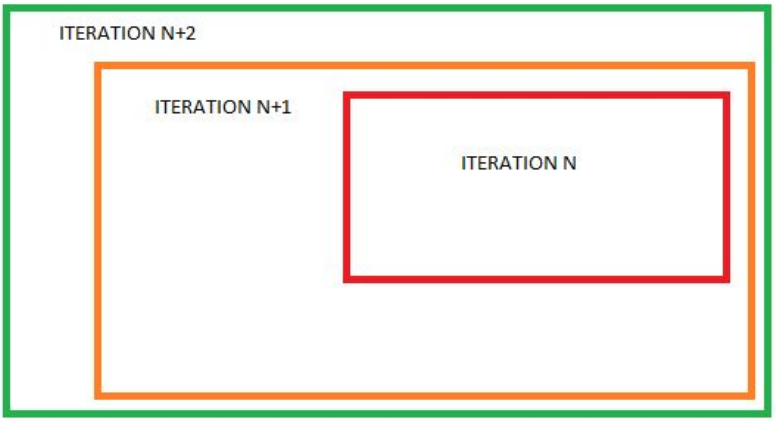
Pour la première itération, nous avons décidé de fixer une liste des premières fonctionnalités à implémenter pour l'interface. Mais avant tout il nous faut réfléchir à l'**architecture du code** que nous allons mettre en place. Cette architecture se veut permettre de factoriser le code un maximum. Rappelons qu'une règle en informatique est d'**éviter la redondance**, et qu'avec le codage d'interface nous allons au devant de beaucoup de répétitions dans le code. Peut-être **réfléchir à un package de classes ou à une classe Tools** (outils).

Nous souhaitons mettre en place en premier la fonctionnalité : **lecture et pause de la vidéo**. Afficher la vidéo, nous semble être l'une des fonctionnalités **atomiques** du logiciel que nous voulons développer. Les fonctionnalités : **convertir une vidéo, extraire une vidéo d'une vidéo, extraire une image d'une vidéo, ajouter une/des vidéos, ajouter une/des images, rogner une vidéo**. Pour résumer pour cette première itération nous souhaitons ajouter l'ensemble des traitements principaux réalisables sur la partie image

d'une vidéo, le son sera le sujet d'une prochaine itération. Ci-dessous **un prototype ABSTRAIT du résultat prévu pour l'itération 1,** il est possible que notre première interface **change selon nos contraintes et nos besoins.**



L'idée étant qu'à chaque nouvelle itération, la nouvelle interface soit construite autour de l'interface de l'itération précédente. Par exemple la construction de l'interface pourrait se faire ainsi : ajouter le JPanel de la vidéo -> ajouter le JPanel des boutons -> ajouter le JPanel des chronologies des vidéos -> ajouter un premier JMenu -> ajouter la JMenuBar des JMenu -> ajouter le JPanel des chronologies des sons -> ajouter le JPanel bibliothèques, etc. Le tout à la manière d'une **progression itérative**. **Chaque itération devra délivrer de nouvelles fonctionnalités directement viables et exécutables en fin d'itération**.



3.3) Liste des fonctionnalités testées

Voici ci-dessous l'ensemble des fonctionnalités que nous avons testés pour nous familiariser à FFmpeg.

Fonctionnalités portant sur le son

- ➢ extraire le son d'une vidéo
- ➢ supprimer le son d'une vidéo
- ➢ ajouter un son a une vidéo
- ➢ extraire un son d'un son
- ➢ augmenter ou diminuer le volume de la bande son d'une vidéo

Fonctionnalités portant sur les vidéos

- ➢ afficher une vidéo
- ➢ convertir d'un format à un autre
- ➢ extraire les images d'une vidéo (en précisant le nombre d'image par secondes à extraire)
- ➢ couper une vidéo
- ➢ rogner une vidéo
- ➢ concaténer des vidéos
- ➢ réduire le nombres d'images par secondes d'une vidéo
- ➢ faire une vidéo avec un fichier image (JPG par exemple) et un son (MP3 par exemple)
- ➢ faire une vidéo à partir de plusieurs images (slideshow)
- ➢ pivoter une vidéo de 90°
- ➢ capturer l'écran de son bureau
- ➢ redimensionnement d'une vidéo (changement resolution)
- ➢ ajouter des sous-titres a une vidéo
- ➢ modifier la taille des sous-titres de la vidéo
- ➢ corriger la synchronisation des sous-titres d'une vidéo
- ➢ pixeliser une vidéo
- ➢ retirer le son d'une vidéo
- ➢ retirer des frames d'une vidéo
- ➢ créer des gifs a partir d'une vidéo
- ➢ créer des gifs a partir d'image
- ➢ retirer certain pixel d'une vidéo comme par exemple ½ pixel
- ➢ couper des parties du son de la vidéo
- ➢ rogner en diagonal la vidéo (du genre la moitié)
- ➢ changer les informations "système" de la vidéo comme :
    - la langue
    - le format (hauteur et largeur de trame)
    - la taille
    - date de création
    - le débit de donné
    - la fréquence par image
    - vitesse de transmission pour l'audio
    - interprète ayant participé(par défaut et souvent rien)

4) Bibliographie

https://www.ffmpeg.org - Site officiel et documentation officiel de FFMpeg.

https://ffmpeg.org/doxygen/3.3/index.html - Documentation sur les librairies de FFMpeg pour les développeurs externes.

https://blog.programster.org/tag/ffmpeg - Articles sur différentes opérations possibles dans FFMpeg.

https://doc.ubuntu-fr.org/ffmpeg - Rapide présentation de FFMpeg et guide d'installation et d'utilisation sous Ubuntu (opérations, multithreads…).

https://fr.wikipedia.org/wiki/FFmpeg - Informations générales concernant FFMpeg.

https://bertiaux.fr/archives/utiliser-ffmpeg-se-prendre-tete - Comment utiliser FFMpeg sans se prendre la tête.

https://www.jcartier.net/spip.php?article36 - FFMpeg par l'exemple.

https://www.hongkiat.com/blog/ffmpeg-guide/ - "A guide to video and audio conversion using FFmpeg".

https://www.linuxtricks.fr/wiki/ffmpeg-la-boite-a-outils-multimedia#paragraph_extraire-l-audio-d-une-videohttps://trac.ffmpeg.org/ - Les commandes principales à connaitre sur FFmpeg.

https://www.developpez.net/forums/d246589/java/general-java/apis/lancer-commande-type-ligne-commande-programme-java/ - Executer des lignes de commandes dans un programme JAVA.