RAPPORT FINAL

Sujet :

Réalisation d'une interface graphique pour un logiciel de traitement vidéo

Tuteur :

OUNI Slim

Professeur accompagnant de la matière :

DEBLED-RENNESSON Isabelle

Membres de l'équipe du projet :

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

Groupe :

S3B

Année 2018-2019

DA SILVA CARMO Alexandre S3B
PALMIERI Adrien
HUBLAU Alexandre
CHEVRIER Jean-Christophe

TABLE DES MATIÈRES

DA SILVA CARMO Alexandre S3B
PALMIERI Adrien
HUBLAU Alexandre
CHEVRIER Jean-Christophe

1. Introduction

a. Objet du document

Ce document présente le projet tutoré que nous avons réalisé en groupe cette année : il réunit une **présentation générale** du sujet de notre projet, une **analyse** de ce sujet, la **liste des fonctionnalités** que nous avons choisi d'implémenter, la **méthodologie** que nous avons **employée** pour les mener à bien, les différences issues de l'évolution du projet ainsi que les difficultés que nous avons rencontrés lors de la réalisation de celui-ci.

b. Présentation du projet

i. Présentation du sujet

On rappelle que le sujet de notre projet consiste en la **réalisation d'une interface graphique**. Cette interface graphique a pour mission d'adapter FFmpeg, un logiciel libre destiné au traitement de flux audio et vidéo utilisable uniquement en ligne de commande. Le but principal est de **rendre accessible FFmpeg aux utilisateurs non informaticiens**, peu coutumiers de l'invite de commande sous Windows, du terminal sous Linux, ou même de tout type de **Shell** (= interpréteur en ligne de commandes).

ii. Comparaison aux solutions existantes

Il existe des solutions qui permettent aujourd'hui de **convertir un fichier** d'un format à un autre. Cela peut s'avérer très utile, notamment lorsque nous souhaitons lire un **fichier audio ou vidéo** sur des **appareils spécifiques tel qu'un décodeur**, un lecteur DVD ou encore un téléviseur, en effet, le **format peut ne pas être compatible avec le lecteur**, notamment si le fichier a été récupéré sur internet : de nombreux formats de conteneurs vidéos et de codecs existent, et ils ne sont pas tous compatibles avec tous les lecteurs.

Des **logiciels** permettant d'effectuer des **conversions existent déjà**, cependant **FFmpeg** est **très puissant** et ne permet pas d'effectuer uniquement cela : il permet également d'**effectuer des traitements** sur les fichiers vidéos : flouter une partie de l'image, insérer une image par dessus la vidéo (watermark), concaténer des vidéos, etc. A ce jour, les **logiciels permettant d'effectuer ce type de traitement sont complexes**, disposent de nombreuses fonctionnalités et sont **payants**.

L'**objectif de notre logiciel** est donc **d'intégrer un système de conversion** ainsi que des **fonctionnalités de base du traitement**, afin de **permettre à l'utilisateur** de convertir ses fichiers et également d'**effectuer facilement des actions** qu'il ne pourrait effectuer qu'en achetant un logiciel qu'il mettrait plus de temps à prendre en main.

c. Membres de l'équipe et répartition des tâches

Nous nous sommes attribués la **responsabilité** de certaines **tâches importantes** dans le projet. Nous avons ensuite par la suite discuté entre nous pour nous répartir les tâches, en sachant qu'être responsable d'une tâche ne nous empêchait pas de travailler sur d'autres partie du projet pour aider nos collaborateurs à corriger des bugs ou à améliorer leur partie.

DA-SILVA CARMO Alexandre :

Responsable de la partie **traitement** et d'une partie de l'**interfaçage** pour la partie traitement. Il a également contribué à certains ajouts graphiques dans le logiciel. Il était également responsable du **trello**.

CHEVRIER Jean-Christophe :

Responsable de l'interfaçage global du logiciel FFmpeg en JAVA ainsi que de l'utilisation des threads. Il a également contribué à certaines modifications graphiques dans le logiciel, ainsi qu'à l'implémentation des notifications, de la gestion des ressources, et de l'historique des réponses de FFmpeg.

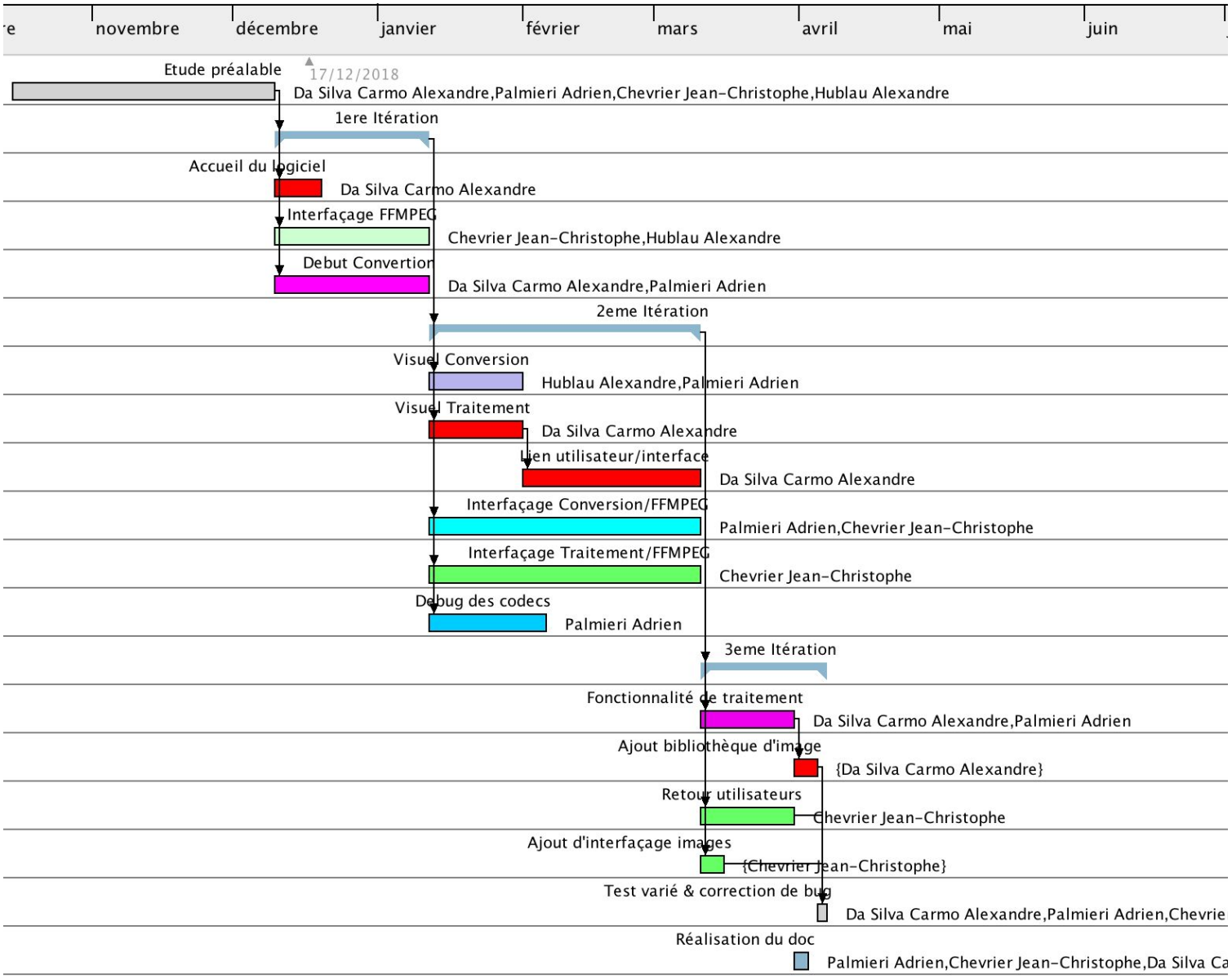
PALMIERI Adrien :

Responsable de la partie **conversion** ainsi que de l'**interfaçage** avec FFmpeg de la partie conversion, notamment pour la gestion des **codecs** et des résolutions. Il a également implémenté certaines fonctionnalités dans la partie **traitement** ainsi que la **gestion des données**.

HUBLAU Alexandre :

Responsable de la personnalisation et du choix des composants graphiques utilisés. Il a implémenté une **surcouche** sur les objets swing classiques, ainsi qu'une partie des fichiers contenant les **métadonnées** et les **traitements** appliqués sur les vidéos.

d. Planning de déroulement du projet



2. Analyse

a. Découpage fonctionnel du projet

i. Méthodologie employée

Nous avons choisi de découper notre logiciel en deux parties : une partie permettant la conversion de fichiers et la seconde le traitement de fichiers. Nous avons choisi de séparer les deux interfaces pour rendre le logiciel plus intuitif, en effet, la partie conversion est plus technique et permet de modifier les métadonnées de la vidéo, tandis que la partie traitement se veut plus graphique et ne nécessite pas de connaissances particulières dans les formats de fichiers.

Nous avons décidé de démarrer le développement par la partie interfaçage, en effet, le lien avec le logiciel FFmpeg doit être fait proprement au préalable pour pouvoir communiquer correctement avec celui-ci et travailler avec lui proprement par la suite. Nous avons ensuite décidé d'implémenter la partie conversion, celle-ci étant moins contraignante à développer dans l'interface graphique pour nous concentrer ensuite sur la partie traitement.

ii. Choix de conception

Nous avons choisi d'utiliser comme technologie le langage informatique JAVA. FFmpeg étant codé en C, nous ne pourrons pas communiquer directement avec ses librairies. Ce pourquoi nous aurons des classes java réalisant un **« interfaçage » entre notre logiciel et FFmpeg**.

Notre logiciel sera découpé en plusieurs packages. Et dans chaque package, des **patrons de conception** seront utilisés. Ces patrons de conception seront adaptés aux fonctionnalités des classes en cause**.** Principal but recherché : **établir une architecture logicielle cohérente et logique, avoir un code factorisé, éviter toute redondance dans les programmes, temps d'exécution optimal et consommer une quantité minimale de mémoire.**

Un choix a été réalisé dans notre équipe de projet, nous avons choisis de réaliser tout le code du logiciel en **ANGLAIS** à l'exception des commentaires. On parle essentiellement ici **des noms des classes, des méthodes et des attributs**, qui seront donc écrits en anglais, afin d'avoir un code homogène.

iii. Sélection des fonctionnalités indispensables

Afin de rendre notre logiciel **utile à l'utilisateur**, sans pour autant **surcharger l'interface** de nombreuses fonctionnalités inutiles (FFmpeg est un logiciel très puissant, il permet d'effectuer beaucoup d'action, et il aurait été handicapant de sélectionner trop de fonctionnalités qui auraient rendue l'interface illisible ou trop complexe pour l'utilisateur), nous avons **sélectionné des fonctionnalités indispensables** pour chaque partie du logiciel.

Concernant les deux parties du logiciel :

- Inspecter les réponses de FFmpeg via un historique de ses réponses
- Etre diriger par des feedbacks via les alertes \ notifications

Pour la partie conversion :

- Changer le format d'un fichier vidéo
- Changer le codec vidéo d'un fichier vidéo
- Changer le codec audio d'un fichier vidéo
- Changer le bitrate vidéo d'un fichier vidéo
- Changer le bitrate audio d'un fichier vidéo
- Changer la résolution d'un fichier vidéo
- Changer le nombre d'images par seconde d'un fichier vidéo
- Changer le taux d'échantillonnage d'un fichier vidéo
- Changer le nombre de canaux audio d'un fichier vidéo

Pour la partie traitement :

- Couper un morceau de vidéo
- Flouter un morceau de vidéo
- Tourner la vidéo dans tout les sens
- Concaténer plusieurs vidéos
- Ajouter une image fixe sur une vidéo (un logo par exemple)

iv. Choix des fonctionnalités supplémentaires

Nous avons sélectionné certaines **fonctionnalités complémentaires** que nous avons choisi d'implémenter dans notre logiciel uniquement lorsque les fonctionnalités indispensables seront implémentées et totalement fonctionnelles.

Pour la partie conversion :

- Ajouter plus de codecs possibles et compatibles

Pour la partie traitement :

- Couper la vidéo chronologiquement
- Modifier toutes les fonctionnalités déjà en place pour pouvoir les faire à des moments précis de la vidéo

b. Évolutions par rapport à l'Étude Préalable

i. Modifications sur le choix des fonctionnalités

Nous avons choisi d'**effectuer des modifications** sur les fonctionnalités que nous souhaitons implémenter dans notre logiciel : nous avons **ajouté certaines fonctionnalités** que nous trouvions intéressantes, **totalement modifié la manière dont nous avions développé certaines fonctionnalités** qui n'étaient pas assez poussées ou pas assez bien réalisées et **supprimé certaines fonctionnalités** dont la charge de travail était trop importante par rapport au résultat obtenu.

ii. Modifications sur la partie traitement

Lors de notre étude préalable, nous souhaitions ajouter une barre de chronologie permettant d'accéder à une frame précise de la vidéo, cependant, certaines **contraintes** en terme de **pré-chargement** des données nous on fait passer cette fonctionnalité en second plan pour nous concentrer en priorité sur les **fonctionnalités les plus utiles**. M.OUNI nous a également dit de changer les fonctionnalité à réaliser pour des fonctionnalité plus simple et plus visuel.

Nous avons également changé le concept de base de la bibliothèque de vidéos dans traitement car M. OUNI a estimé que l'utilisateur n'avait pas besoin de cette fonctionnalité pour la partie traitement, de plus le développement de cette fonctionnalité nous aurait compliqué la tâche, la bibliothèque n'étant pas nécessaire pour effectuer un traitement. Donc nous avons converti cette bibliothèque vidéo en un bibliothèque d'images pour que l'utilisateur puisse les incruster à sa convenance.

iii. Amélioration de l'interfaçage

Lors de l'étude préalable, nous avions uniquement 3 classes qui géraient la communication avec FFmpeg : FFmpegRuntime, UserRequests et SystemRequests. Cela n'était pas suffisant : la **communication** avec le logiciel ffmpeg était **trop "directe**" et ne permettait pas de **gérer proprement plusieurs requêtes** à la suite, ne permettait pas d'avoir de **retours sur les erreurs** lors des conversions et posait des **problèmes sur certains systèmes d'exploitation**. Nous avons donc **revu l'interfaçage** afin de l'**améliorer et le rendre plus poussé** pour permettre de répondre correctement aux besoins de notre logiciel.

iv. Amélioration de l'interface utilisateur

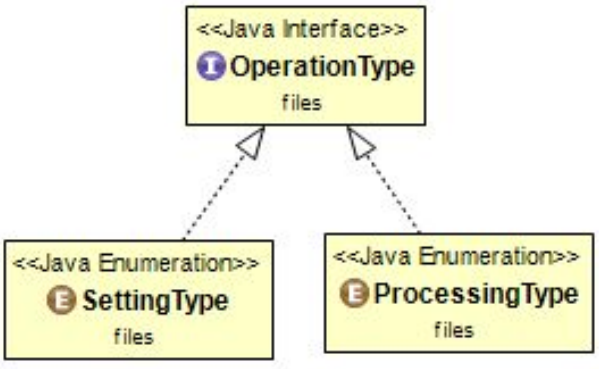
Nous avons choisi de rendre l'interface utilisateur plus **intuitive**, en mettant en avant les points les plus importants, notamment dans la **partie conversion** qui contient **beaucoup d'informations** sur les fichiers importés et qui permet à l'utilisateur de modifier beaucoup de paramètres en sortie. L'**interface précédente** était **assez floue**, l'utilisateur avait du mal à distinguer les paramètres en entrée de ceux en sortie et l'interface n'était pas assez intuitive. Nous avons donc **amélioré la mise en page de l'interface**, utilisé des couleurs et du **HTML/CSS** dans les composants et utilisés des composants graphiques **Swing** modifiés afin de rendre l'interface plus lisible et plus claire pour l'utilisateur.

c. Diagrammes de classes principaux

Les diagrammes de classe ci-dessous ne **présentent uniquement** que certaines **parties essentielles** du logiciel. De nombreux éléments ne pourront pas être évoqués dans un soucis de **limiter la complexité** du rapport. Ces diagrammes de classes ne sont donc pas dans leur entièreté exhaustif. De plus certains de ses diagrammes ne présentent pas non plus le tous les détails portant sur les classes.

Diagrammes de classes de la famille des types d'opérations :

(package src.files.énumérations)

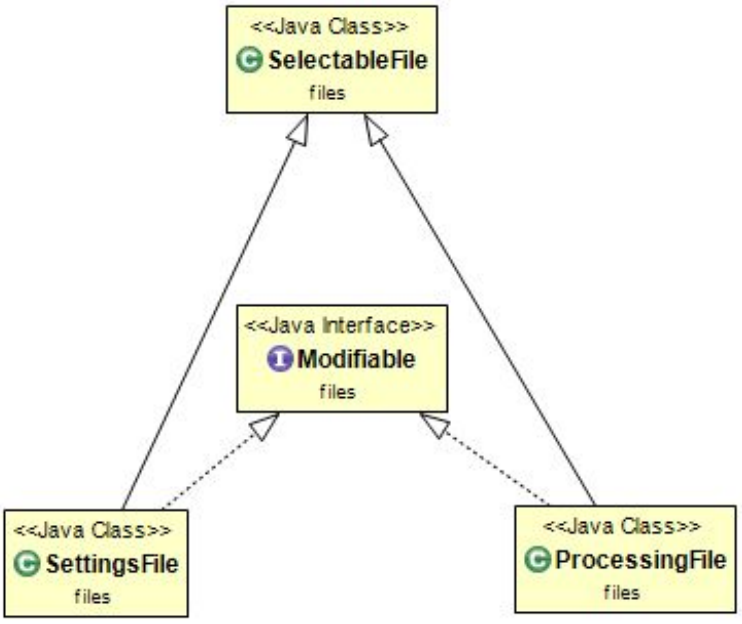


Une type d'opération est une clé dans nos tables de données. (HashMap) Un type d'opération peut-être **un type de métadonnée modifiable** (SettingType) ou **un type de traitement applicable** (ProcessingType) sur une vidéo ou un son.

DA SILVA CARMO Alexandre S3B
PALMIERI Adrien
HUBLAU Alexandre
CHEVRIER Jean-Christophe

Diagrammes de classes de la famille des fichiers :

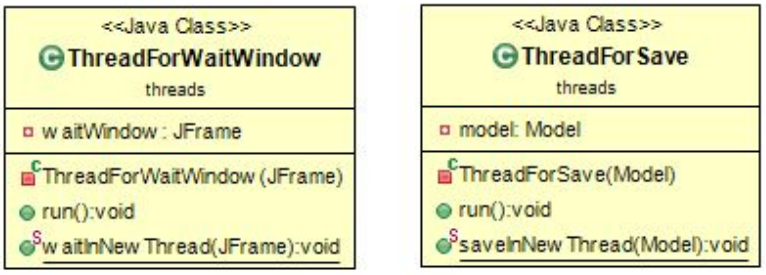
(package src.files.files)



Un SelectableFile contient un fichier (File) et une table (Map). Le fichier est une vidéo ou un son. La table se contente d'être la structure de données qui **enregistre les modifications à réaliser** sur le fichier depuis FFmpeg quand l'utilisateur lance l'export du fichier. La table associe à u**ne clé de type OperationType une valeur de type String**.

Diagrammes de classes des threads :

(package src.threads)



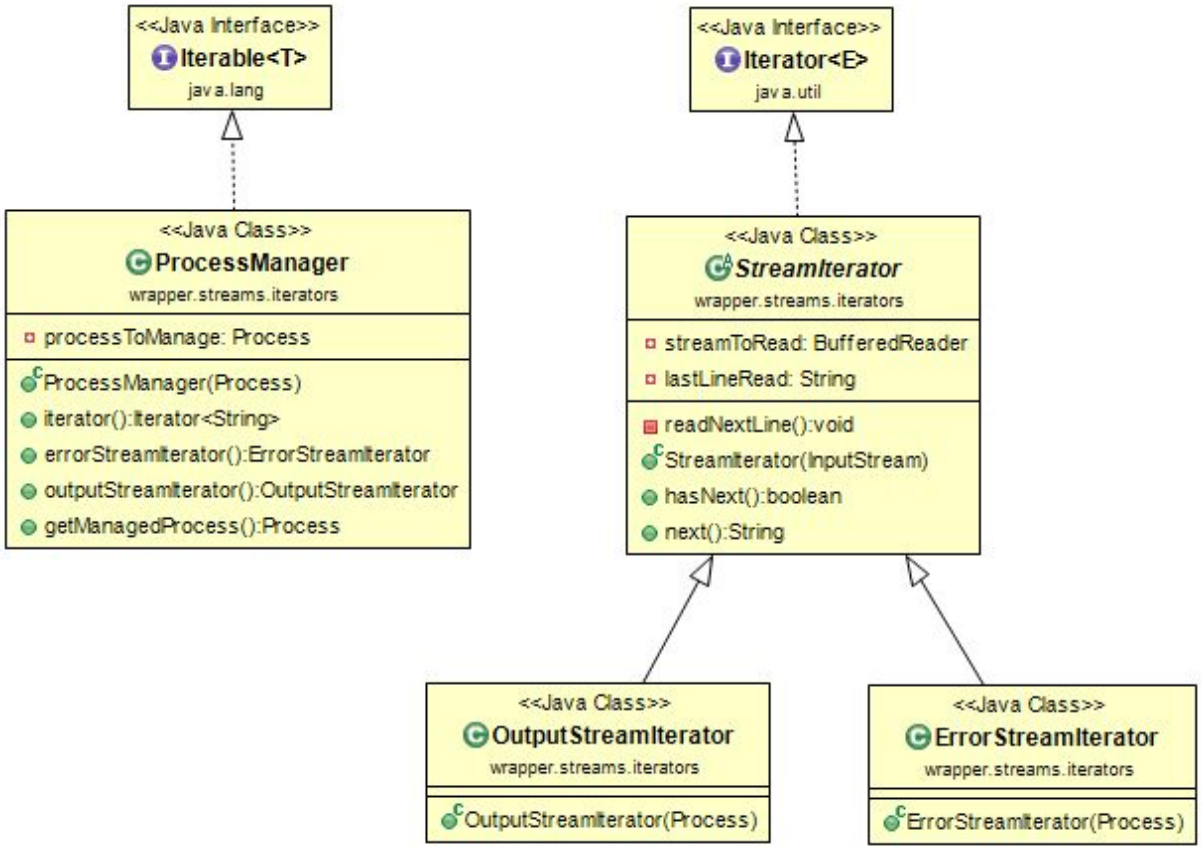
Le fonctionnement en interne des threads nous obligeraient à expliquer tout l'environnement de classes qu'ils manipulent, nous n'irons donc pas dans le détail. Il est à retenir dans l'essentiel que les threads nous permettent de **faire tourner en parallèle JAVA et FFmpeg**, et de **conserver une ainsi une interface stable**.

DA SILVA CARMO Alexandre S3B
PALMIERI Adrien
HUBLAU Alexandre
CHEVRIER Jean-Christophe

Diagrammes de classes des itérateurs sur les flux de réponses de FFmpeg:

(package src.wrapper.streams.iterators)

*

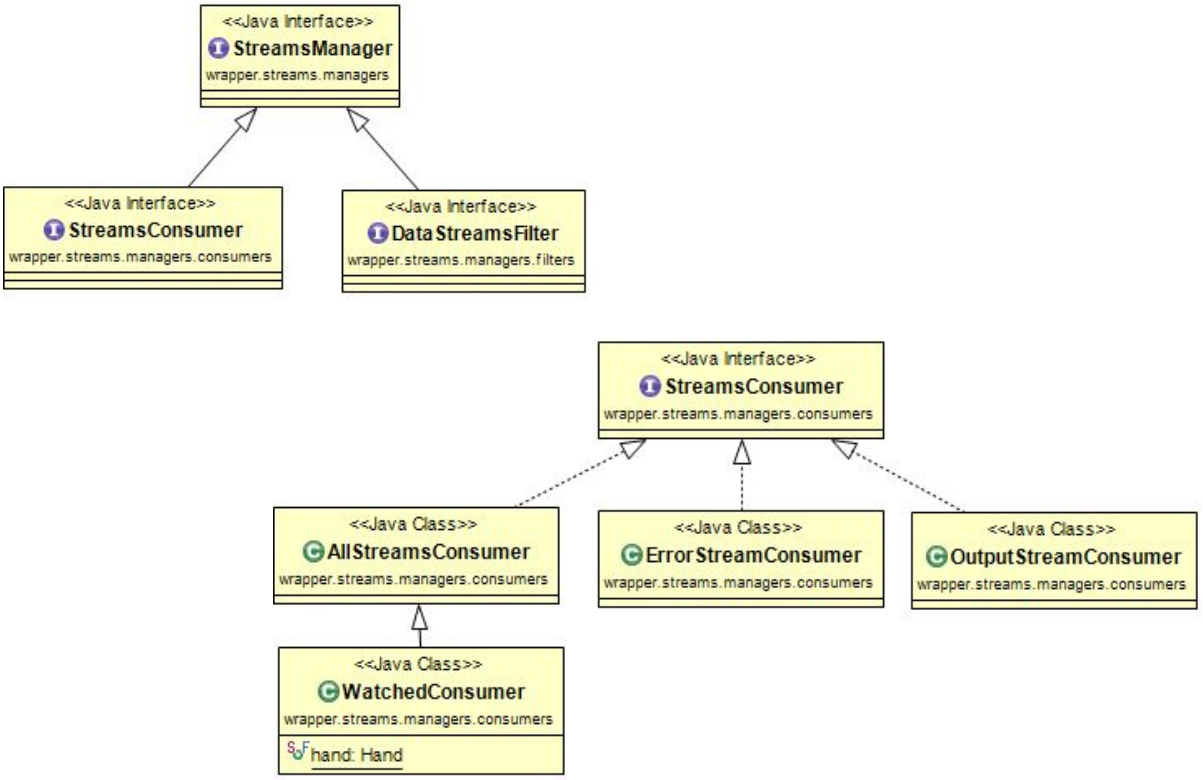


Nous avons réalisé ces itérateurs pour **simplifier le parcours des flux** qui nous sont retournés par FFmpeg. Ce patron nous a permis de rendre le parcours des flux de réponses de Ffmpeg beaucoup plus intuitif.

DA SILVA CARMO Alexandre S3B
PALMIERI Adrien
HUBLAU Alexandre
CHEVRIER Jean-Christophe

Diagrammes de classes de la famille des gestionnaires de flux :

(package src.wrapper.streams.managers)



Ce diagramme présente la grande famille des gestionnaires de flux qui comportent **les filtres sur les flux** et **les consommateurs de flux**. Il faut noter que que dans les deux cas : filtrage ou consommation, **le contenu des flux est toujours sauvegardé** dans fichiers uniques horodatés. La différence se trouve dans le cas des filtres où on analyse le contenu d'un flux en plus de le sauvegarder.

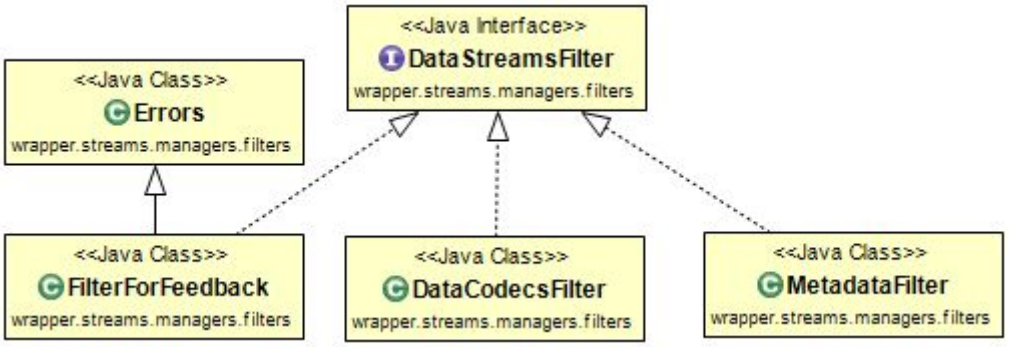
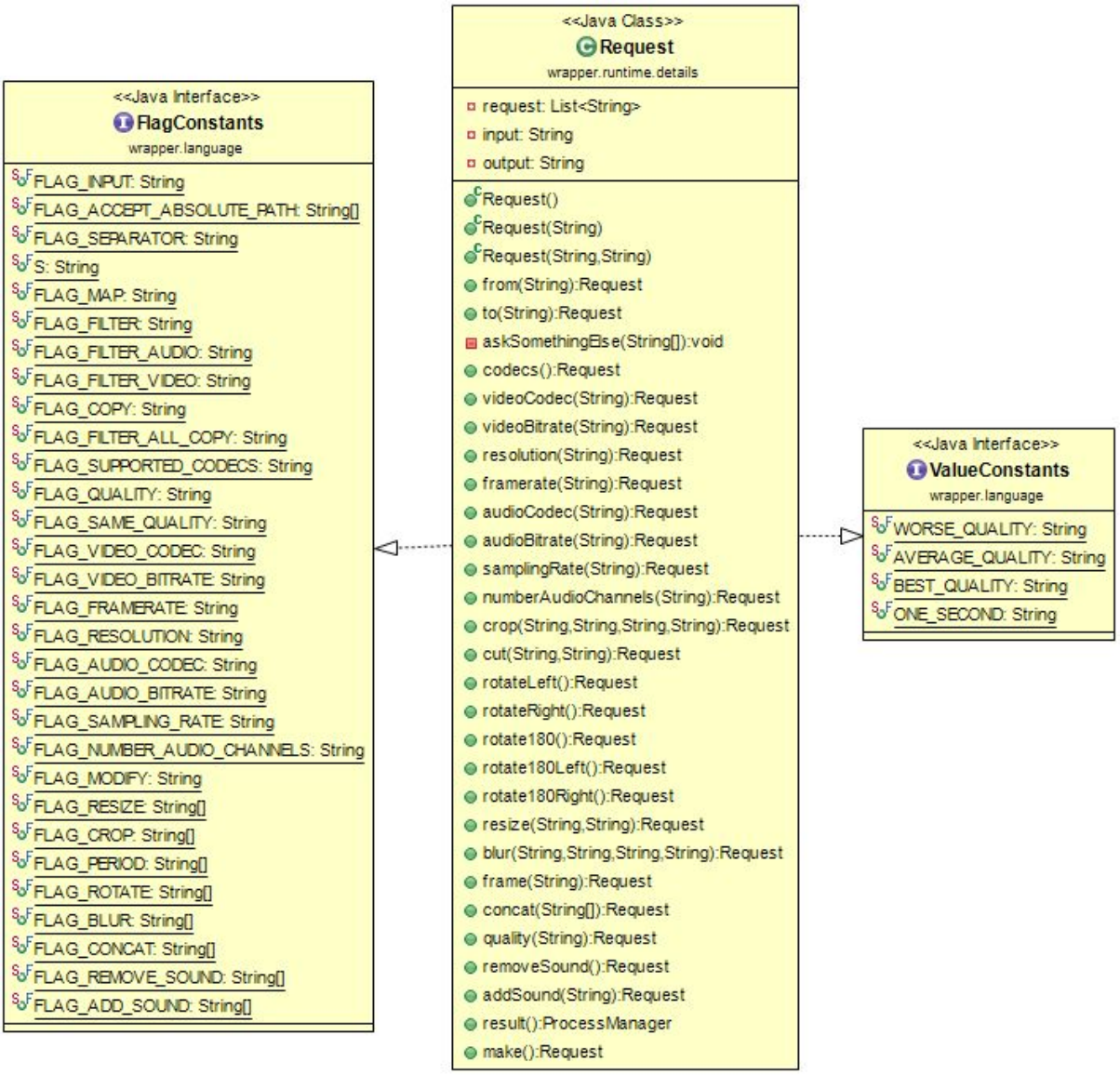


Diagramme de classes de la réalisation des requêtes :

(package src.wrapper.language et src.wrapper.runtime.details)

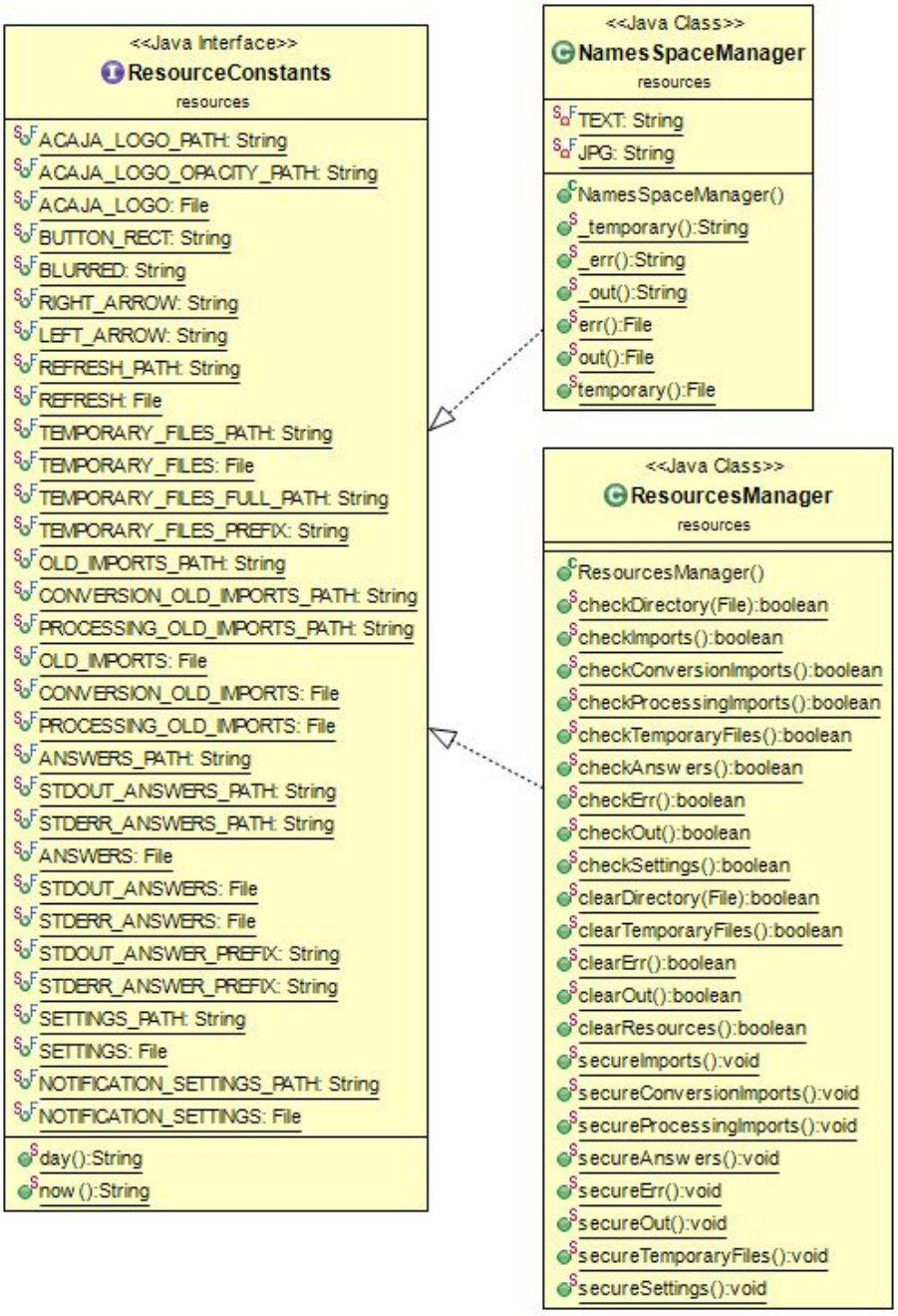


La classe Request a pour rôle d'être un **outil intuitif de construction de requêtes** FFmpeg. Pour cela elle utilise en interne les **flags et valeurs utilisés** par FFmpeg référencés dans des interfaces. Request propose deux méthodes d'exécution : result(), qui exécute la requête et **retourne les flux de réponses** de FFmpeg dans un ProcessManager, et make() qui exécute la requête et **se charge en interne de consommer les flux de réponses** en les sauvegardant dans des fichiers horodatés. **La majorité des méthodes de**

Request retourne this pour pouvoir enchaîner les méthodes les unes sur les autres autant que l'envie nous en prend. **L'interfacage fonctionne avec le type String pour l'essentiel**.

Diagramme de classes de la gestion des ressources :

(package src.resources)

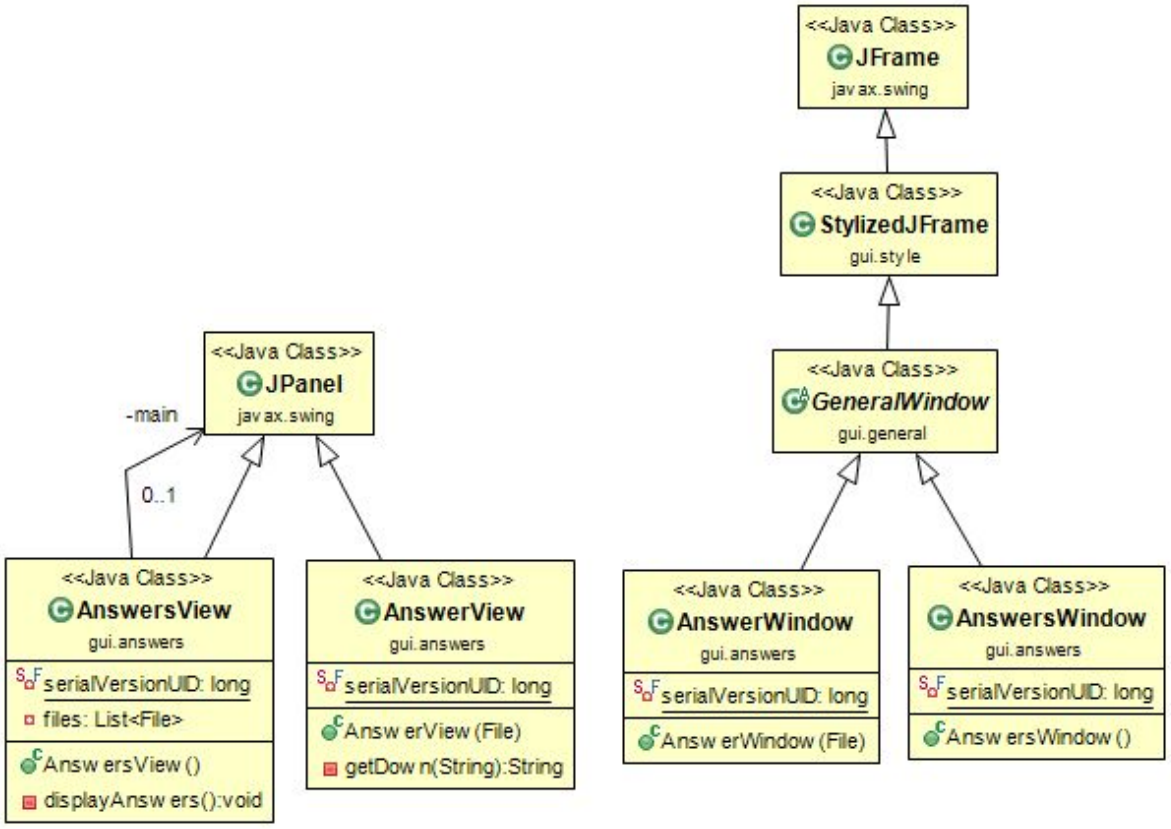


La classe NameSpaceManager nous sert à **attribuer des noms de fichiers neufs et uniques** dans différents répertoires. ResourcesManager nous permet quant à elle de **vérifier/rétablir** une ressource, **supprimer** une ressource, **lever des exceptions** si une

ressource est introuvable et impossible à rétablir. Enfin ResourceConstants sert à **référencer les ressources.**

Diagramme de classes de l'historique des réponses de FFmpeg :

(package src.gui.answers)

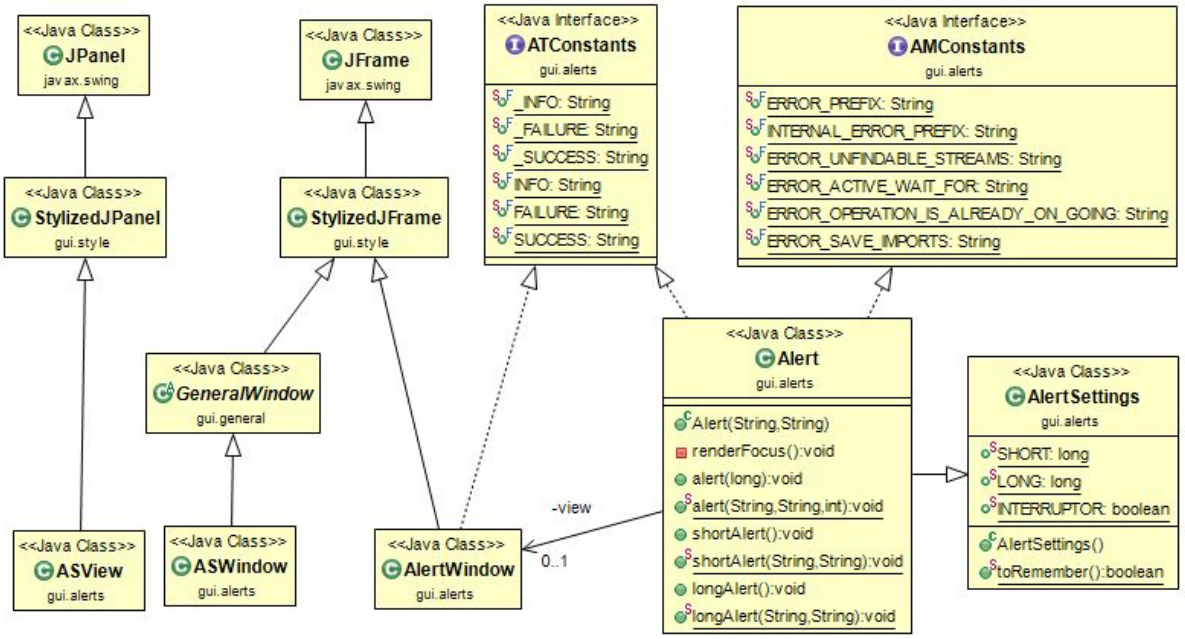


Ces classes servent à construire les fenêtres respectives de l'historique des réponses de FFmpeg, et de chaque réponse individuellement. L'affichage d'une réponse se fait par l'intermédiaire d'un **JEditorPane**, le texte de la réponse y est injecté dedans v**ia du balisage HTML / CSS**, une balise <style> y est injecté entre autre pour **créer des thèmes de couleur** dans l'affichage de la réponse.

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

Diagramme de classes des alertes / notifications:

(package src.gui.alerts)



L'interface ATConstants (avec AT = Alert Type) **référence les 3 grands types d'alertes**.

L'interface AMConstants **référence des messages divers d'alertes.**

La classe AlertWindow **construit l'alerte d'un point de vue graphique**. à savoir la fenêtre d'alerte.

La classe Alert permet d'**afficher une alerte limitée dans le temps**.

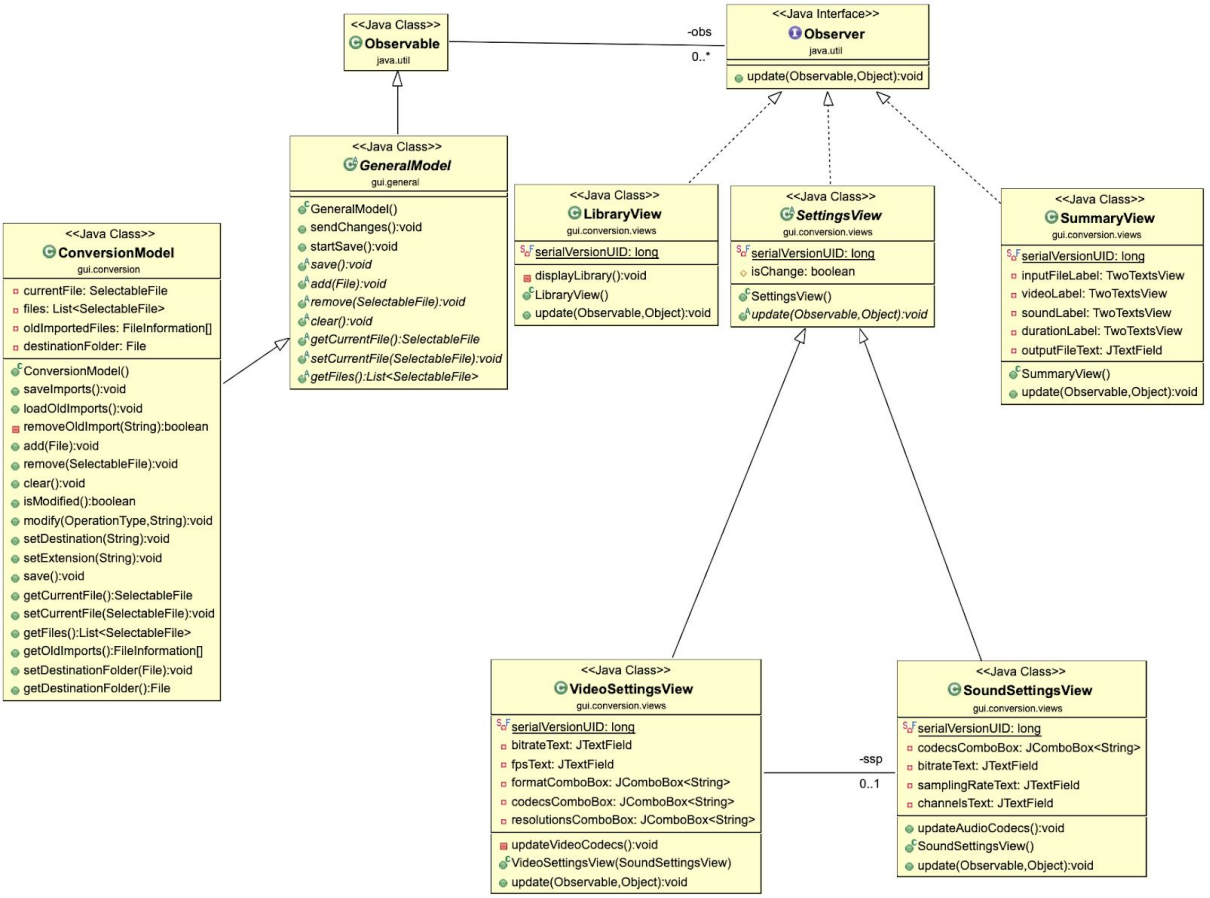
La classe AlertSettings **référence, charge, et enregistre les préférences de l'utilisateur** vis à vis des alertes. Un **fichier .ini** sauvegarde .les préférences de l'utilisateur. Il sert à **conserver les préférences de l'utilisateur** à la fin de l'exécution du logiciel.

Les classes ASView et ASWindow (avec AS = Alert Settings) permettent de proposer **une interface de modifications des préférences**.

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

Diagrammes de classes du MVC de la partie Conversion :

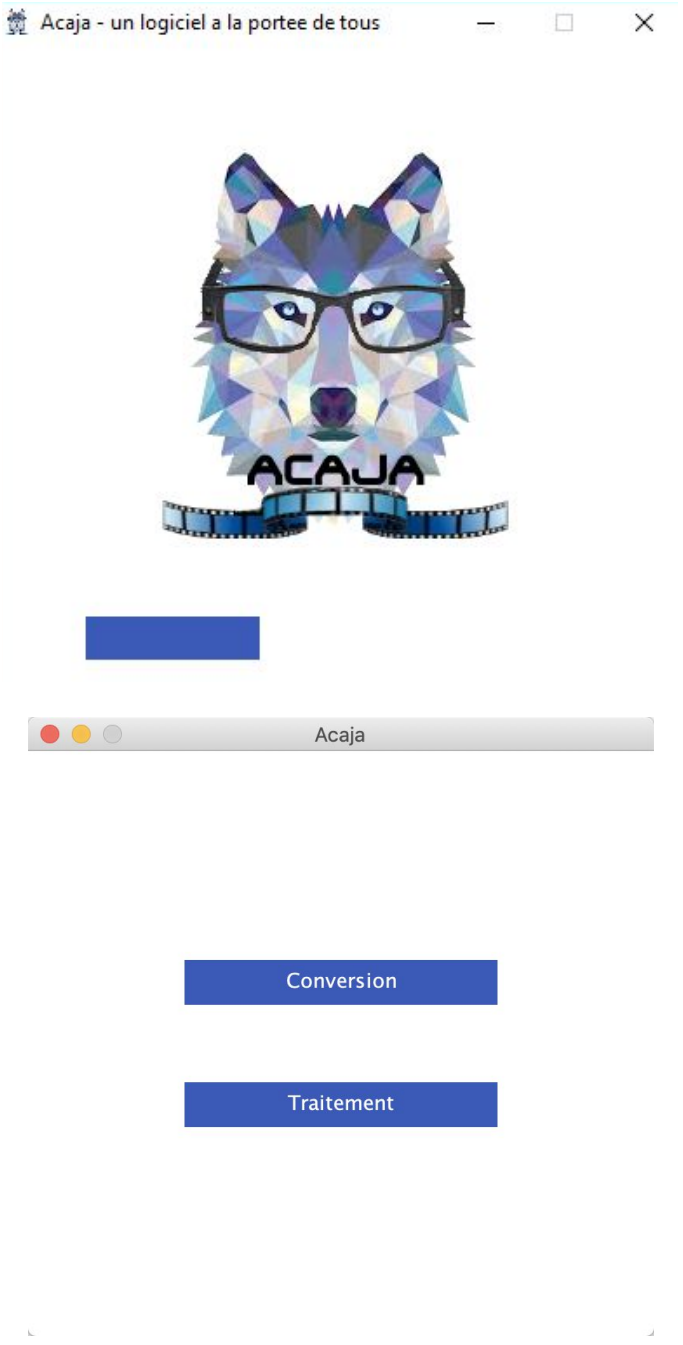
(packages src.gui.conversion et src.gui.conversion.views)



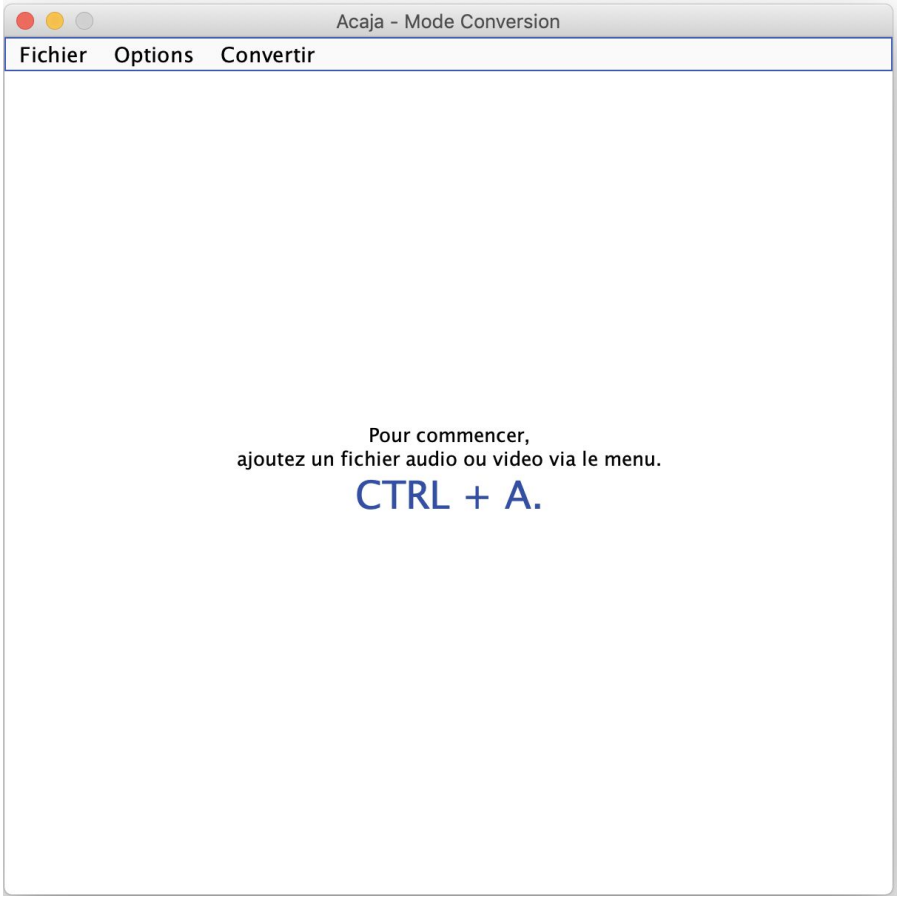
Dans la **partie conversion**, notre **modèle re**présente la **liste des fichiers de type SettingsFile importés** dans le logiciel ainsi que **leurs paramètres de sortie**. Notre modèle hérite d'un **modèle général** (car nous utilisons un modèle différent pour la partie traitement) qui lui même hérite de **Observable**. Nos **vues** implémentent la classe **Observer** et se mettent à jour lors d'un changement quelconque sur le modèle.

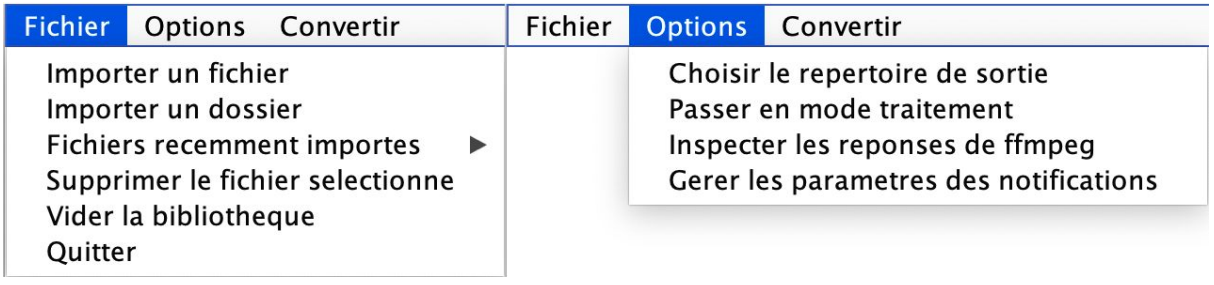
e. Maquettes finales du projet

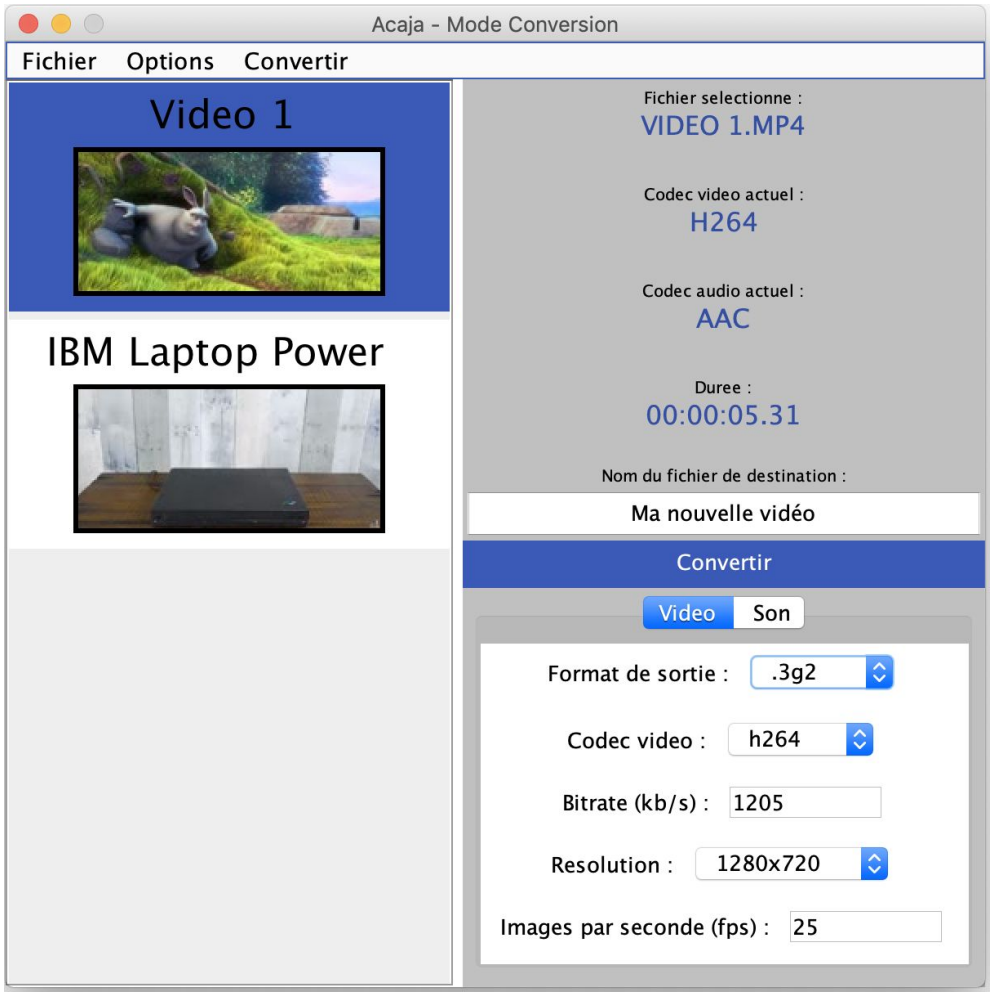
Fenêtre d'accueil



Fenêtre de conversion

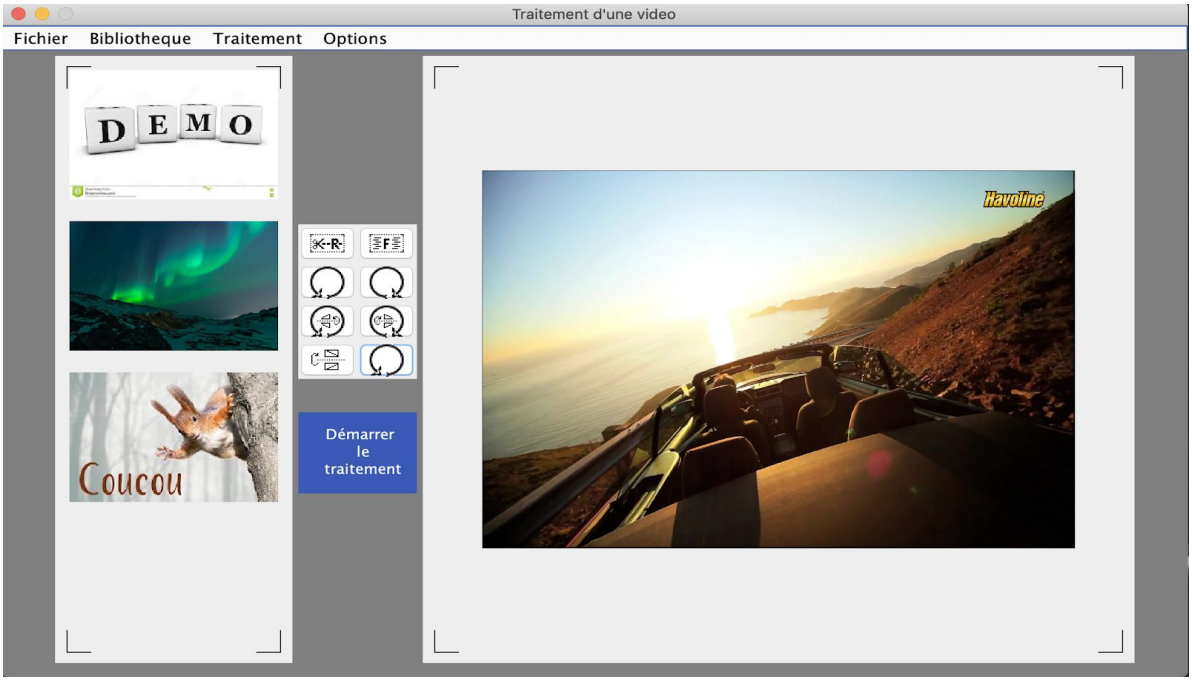


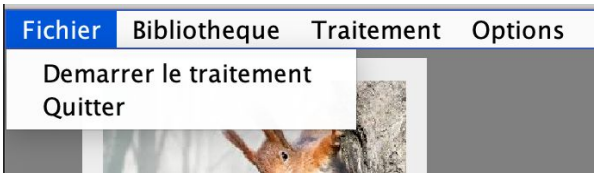


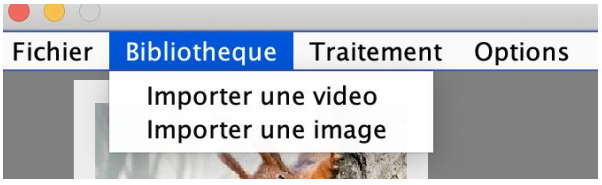

Acaja - Mode Conversion
Fichier
Options
Convertir
Video 1
IBM Laptop Power
Fichier selectionne :
VIDEO 1.MP4
Codec video actuel :
H264
Codec audio actuel :
AAC
Duree :
00:00:05.31
Nom du fichier de destination :
Ma nouvelle vidéo
Convertir
Video
Son
Format de sortie :
.3g2
Codec video :
h264
Bitrate (kb/s) :
1205
Resolution :
1280x720
Images par seconde (fps) :
25

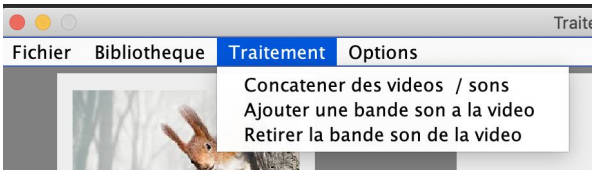
DA SILVA CARMO Alexandre S3B
PALMIERI Adrien
HUBLAU Alexandre
CHEVRIER Jean-Christophe

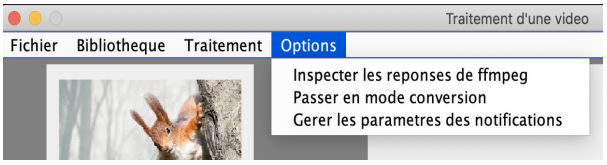
Fenêtre de traitement :

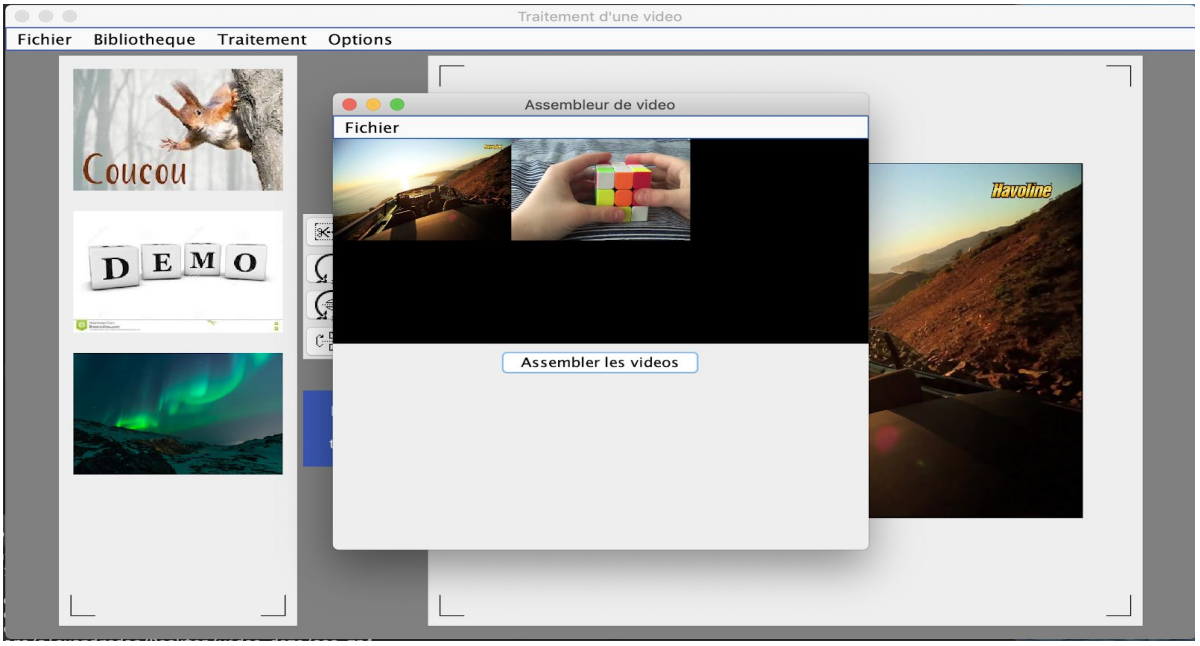




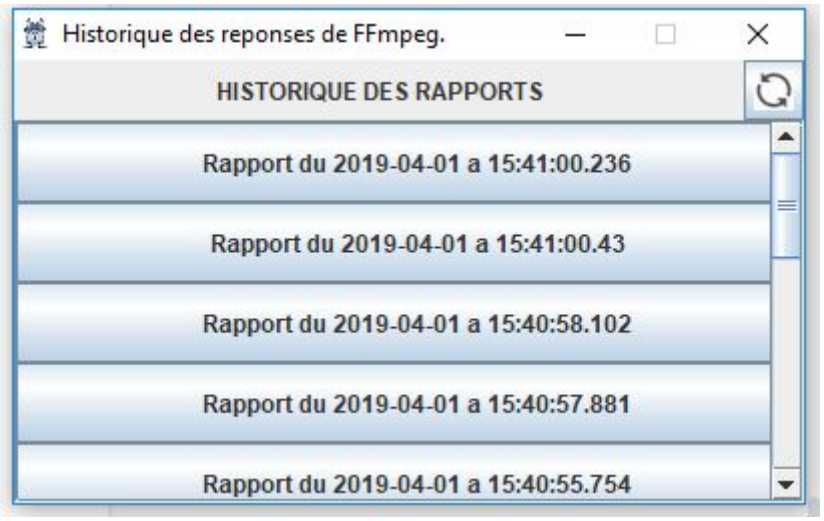


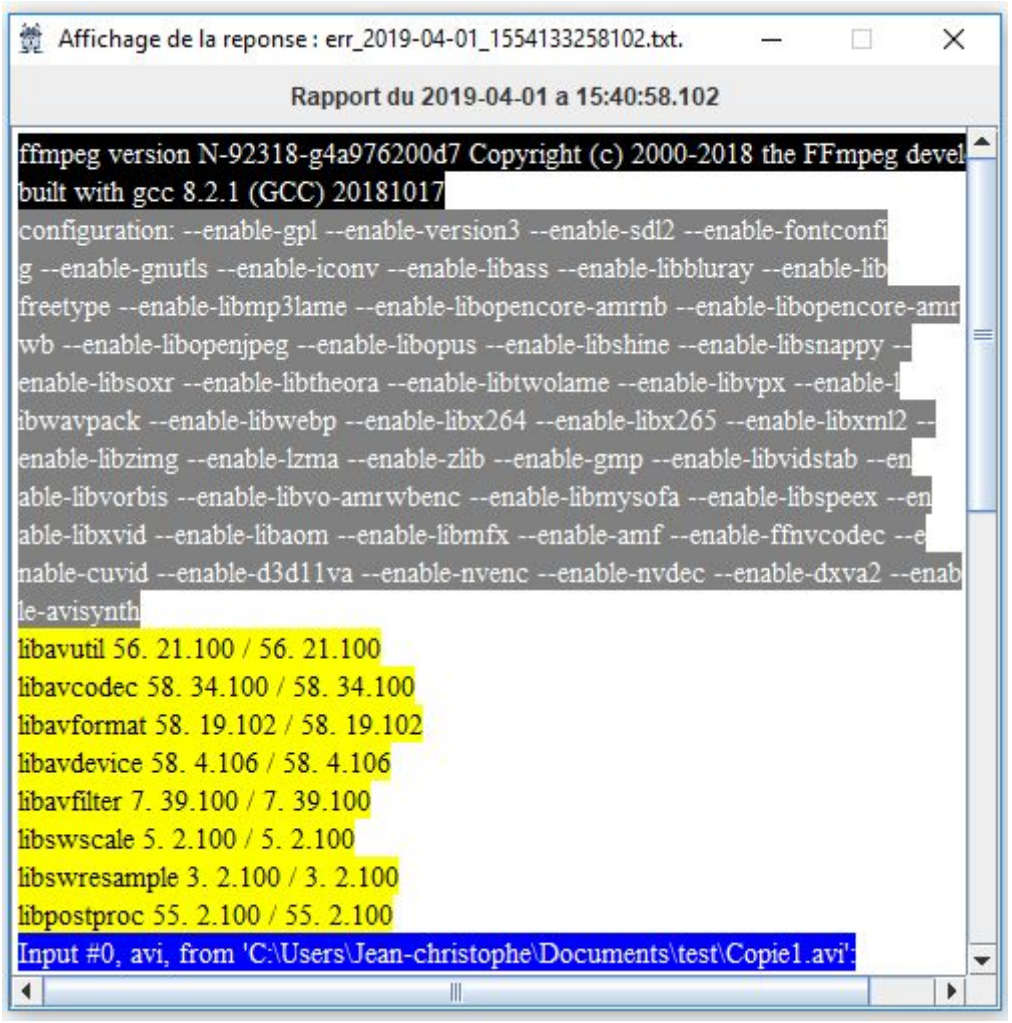


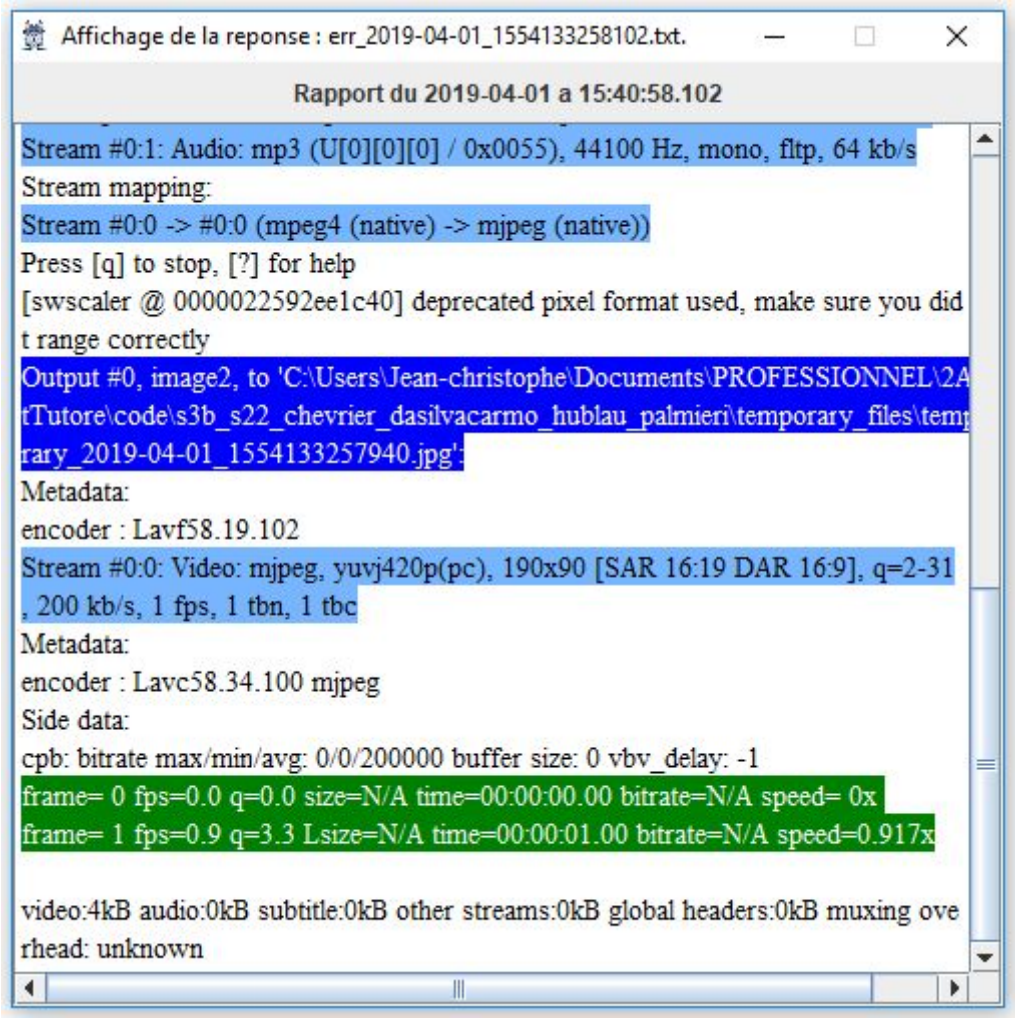




Fenêtre “ inspecteur “ des réponses de ffmpeg :







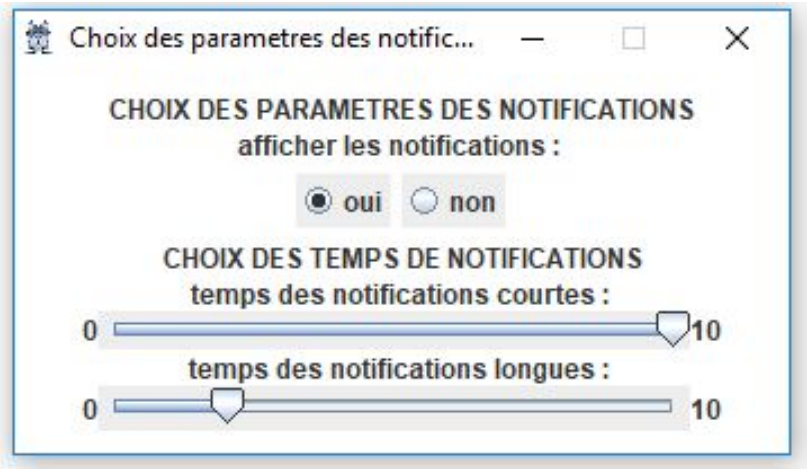
Filtrage des données et **coloration des données par thèmes de données :**

- **version et données** sur FFmpeg : noir ;
- **options** : gris ;
- **librairies de FFmpeg** : jaune ;
- **chemins et noms** sur les fichiers d'entrée et de sortie : bleu foncé ;
- **métadonnées** sur les fichiers d'entrée et de sortie : bleu clair ;
- **évolution de la requête** FFmpeg : vert ;
- **message d'échec** de la requête : rouge ;

Fenêtres de notification :

3 thèmes de couleur pour les notifications :

- thème INFO bleu ;
- thème FAILURE rouge ;
- thème SUCCESS vert ;



3. Réalisation

a. Architecture du logiciel

i. Patrons de conception utilisés

- Patron **Iterator-Iterable** pour le parcours de flux de réponses de FFmpeg. (package src.**wrapper.streams.iterators**) ;
- Patron **Factory** pour l'usage des threads (package src.**threads**) ;
- Patron **MVC** (package **src.gui.conversion** et src.**gui.processing** et classe src.**gui.general.GeneralModel**) ;
- Patron **Stratégie** (**tous** les packages) ;
- Patron **Producteur-Consommateur** (package **src**.**threads** et classes src.**resources.Hand** et src.**wrapper.streams.managers.consumers.WatchedConsumer**).

ii. Utilisation des threads

Lors de l'étude préalable, nous nous doutions qu'il faudrait utiliser les threads en Java pour éviter que les appels au logiciel FFmpeg figent l'interface et que l'utilisateur pense que le logiciel est planté lorsqu'une opération sur FFmpeg serait en cours. Nous n'avions pas implémenté ces threads lors de la première itération et cela posait problème. En effet, sans les **threads**, si FFmpeg plante ou si la commande envoyée au logiciel n'aboutit pas, la fenêtre plante aussi, et lorsque notre logiciel attendait une réponse de FFmpeg, il était impossible de l'utiliser. Nous avons donc mis en place des threads afin de pouvoir continuer à utiliser notre logiciel lorsque FFmpeg travaille et également afin de pouvoir **lancer plusieurs instances** de FFmpeg simultanément si l'utilisateur le souhaite.

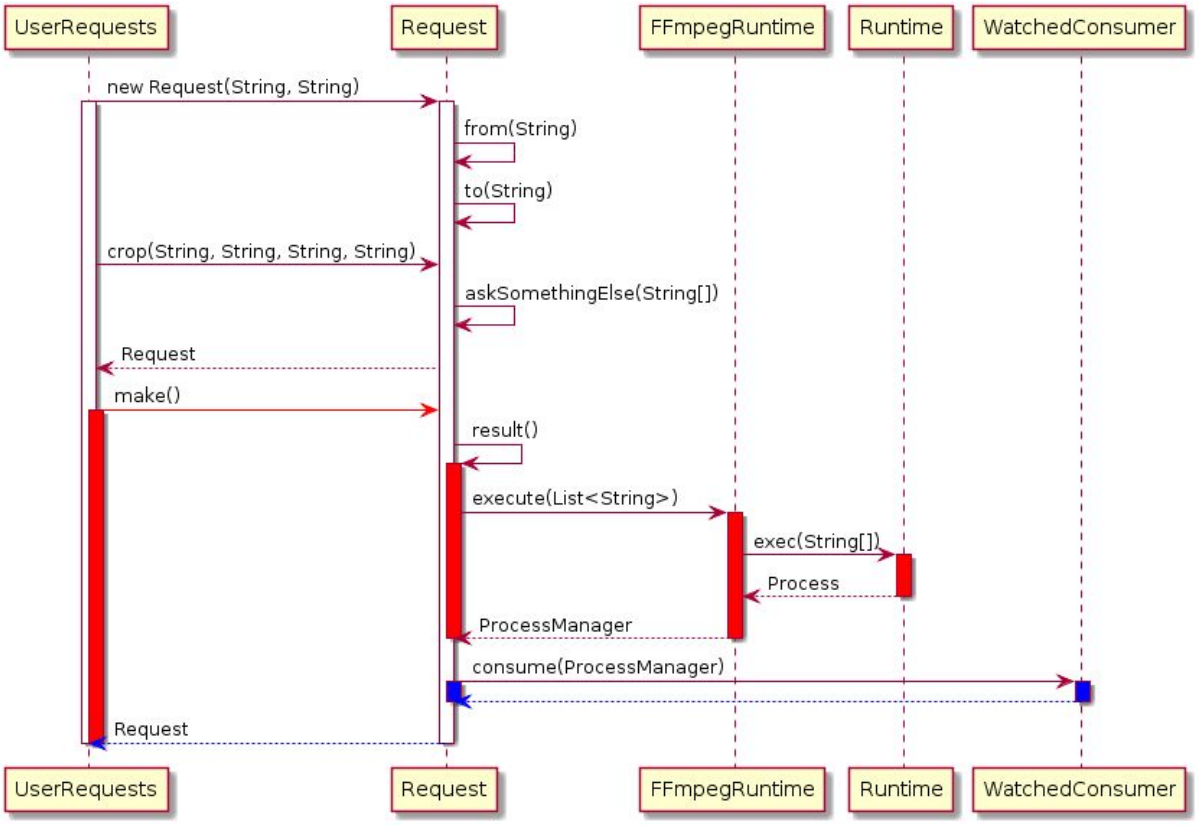
iii. Partie graphique

Concernant la partie graphique, nous avons réalisé des composants Swing personnalisés, qui héritent des composants **Swing** classiques mais dont certains attributs ont été modifiés pour créer un thème unique et donner une identité à notre logiciel. Nous avons optimisé la mise en page des fenêtres de sorte à les rendre les plus user-friendly possible tout en donnant à l'utilisateur les informations dont il a besoin pour travailler sur ses fichiers.

iv. Partie interfaçage

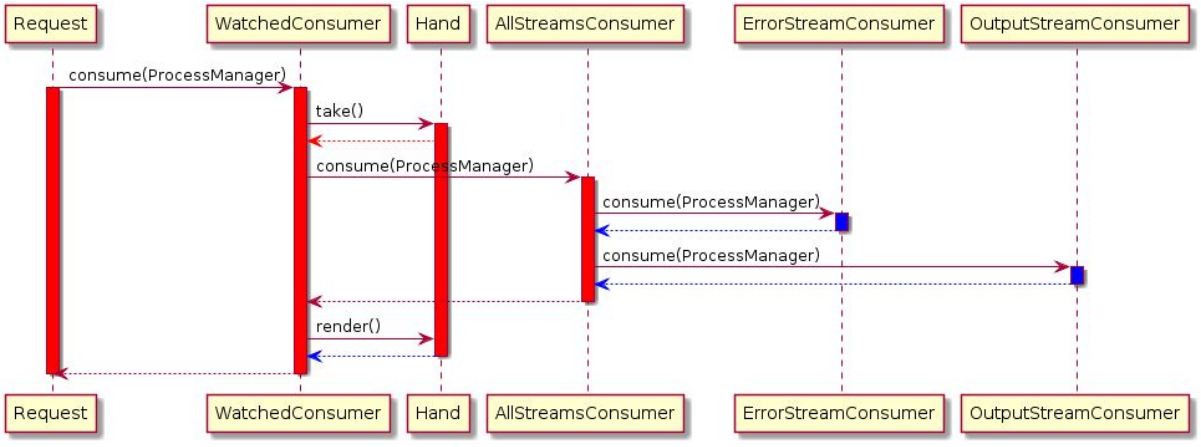
Parler de l'interfacage pourrait prendre des dizaines et des dizaines de pages, au lieu de ça nous vous proposons des diagrammes de séquences pour faciliter sa compréhension.

Diagrammes de séquences de la réalisation d'une requête



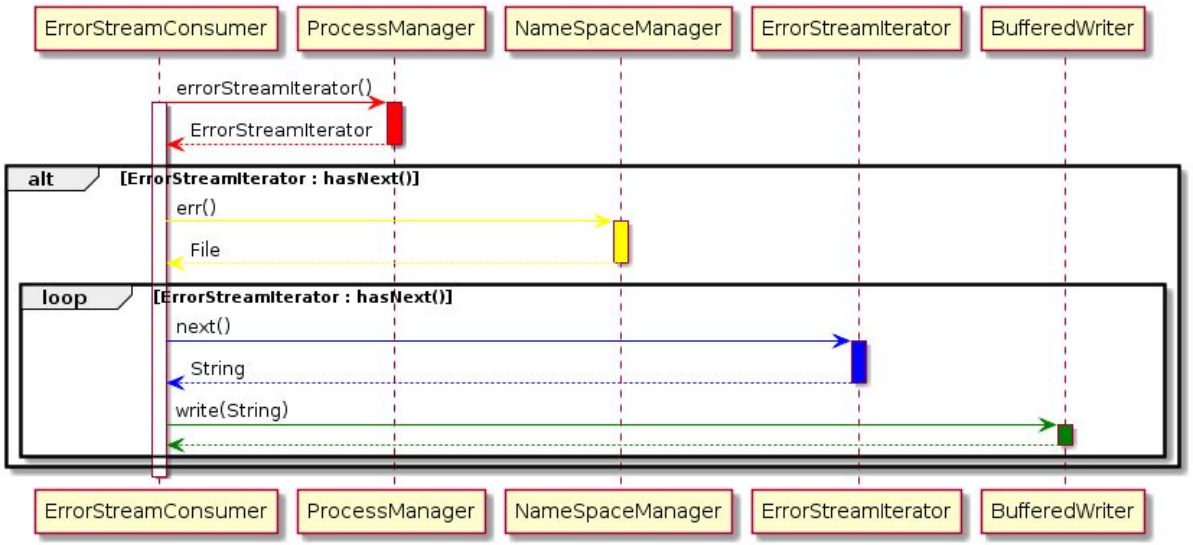
Ce diagramme montre comment se déroule une requête **de sa construction à son exécution.**

Diagrammes de séquences de la consommation "surveillée" des flux (global)



Ce diagramme montre comment se déroule la **consommation "surveillée"** des flux d'un point de vue global.

Diagrammes de séquences de la consommation des flux (concret).



Ce diagramme décrit la manière dont les flux d'erreur de FFmpeg sont consommés via la méthode consume() de la classe **ErrorStreamConsumer**. La méthode consume() appelle la méthode **errorStreamIterator()** sur le **ProcessManager** (Iterable) qui lui renvoie un **ErrorStreamIterator** (Iterator) qui est un itérateur sur le flux d'erreur à consommer, représenté en rouge.

Si le flux du **StreamIterator** n'est pas vide, elle appelle la méthode err() de la classe **NameSpaceManager** qui génère un File, dont le nom est **unique et horodaté**. Ce File sera le fichier de sauvegarde du flux à consommer.

Tant que le flux de l'itérateur contient des lignes à consommer, ces lignes sont écrites dans le fichier. Le tampon mémoire contenant le flux est progressivement vidé, et de fait libère la mémoire. Le fichier à la fin contient toute la réponse de FFmpeg sauvegardée.

Ainsi on vide le tampon mémoire où est stocké le flux d'erreur de FFmpeg et conservons ce flux dans un fichier. Consommer un flux si il n'est pas utilisé est très important. Sans cela il y a de potentiels risques d'interblocage entre JAVA et FFmpeg dû aux tampon mémoire bloqué par des flux stockés dedans non consommés.

De plus ces fichiers de sauvegardes nous servent ensuite pour étudier a posteriori l'état de la requête FFmpeg à l'origine des flux.

b. Tests de validation

i. Méthodologie employée pour les tests

Pour tester notre code, nous avons mis en place des classes de tests afin de vérifier que nos méthodes ne renvoyaient pas de valeurs incorrectes. Nous avons également testé chacune des fonctionnalités sur les différents systèmes d'exploitations possibles afin de vérifier que tout était en ordre.

Concernant la partie conversion, FFmpeg propose énormément de codecs différents, tous ne sont pas compatibles avec tous les formats, et certains comme le h.263 n'acceptent pas toutes les résolutions. Il nous a fallu mettre en place un document nous permettant d'indiquer les erreurs que nous rencontrions lors de certaines conversions afin de par la suite corriger les erreurs ou supprimer les codecs peu utilisés et trop contraignants.

c. Difficultés rencontrées

i. Difficultés liées à FFMPEG

FFmpeg est un logiciel très puissant et contient de nombreuses fonctionnalités, cependant celui-ci s'exécute uniquement en **ligne de commandes**. En utilisant une interface graphique, le risque est d'autoriser à l'utilisateur un enchaînement d'actions sur l'interface qui ne pourraient pas être exécutées en une seule ligne de commande et qui **nécessiteraient plusieurs opérations** (commandes) dans FFmpeg. Cependant, l'architecture de notre logiciel ne permettant pas cette utilisation précise, il a fallu sélectionner les actions que l'utilisateur peut ou ne peut pas faire une fois une action effectuée, notamment dans la partie traitement.

ii. Contraintes liées aux composants graphiques

Nous avons fait le choix d'utiliser **Swing** car il s'agissait de librairies que nous connaissions. Cependant, nous avons vite constaté les limites de ces librairies et avons dû créer des composants personnalisés et utiliser du **HTML/CSS dans certains cas pour embellir l'interface**.

Nous avons également dû tester notre interface graphique sur différentes résolutions d'écrans afin d'être sûrs que le rendu du logiciel soit correct chez tous les utilisateurs.

iii. Contraintes liées au multi-plateforme

L'un des avantages de notre logiciel comparé aux autres est le fait qu'il soit **multi-plateforme** : il peut aussi bien fonctionner sous Windows, Linux, que macOS. Cependant, FFmpeg ne s'intègre pas de la même manière sur les **différents systèmes** et cela nous a valu quelques soucis, notamment pour l'exécution de FFmpeg et également pour la manière de **traiter les requêtes** : sous Windows par exemple, le système pouvait très bien interpréter les commandes envoyées sous la forme de chaînes de caractères tandis que sous Linux et macOS il fallait obligatoirement que les arguments des commandes soient renvoyées sous la forme d'un tableau de chaîne de caractères, chaque élément du tableau représentant un argument ou le paramètre d'un argument.

4. Conclusion

Au terme de cette étude, nous avons appris avant tout à utiliser le logiciel FFmpeg, qui, par ailleurs, est un logiciel vraiment très **complet**. Son utilisation en **lignes de commandes** n'est vraiment pas adaptée à un **utilisateur lambda**, ce qui nous a pris un petit peu de temps pour nous adapter.

Ensuite pour ce qui est du langage de programmation JAVA, on peut en dire que c'est un langage lourd en terme de performances, ce qui combiné avec les temps de traitements de FFmpeg rend d'autant plus lent le programme.

Pour finir le projet nous a également obligé à nous confronter au travail de groupe et à tous ces problèmes classiques et pourtant si inévitables parfois.