Domain Specific Languages: CinEditorML

DOMAIN SPECIFIC LANGUAGES

BERTIN Loïc
FANTAUZZI Virgile
LADORME Guillaume
VIALE Stéphane

Introduction

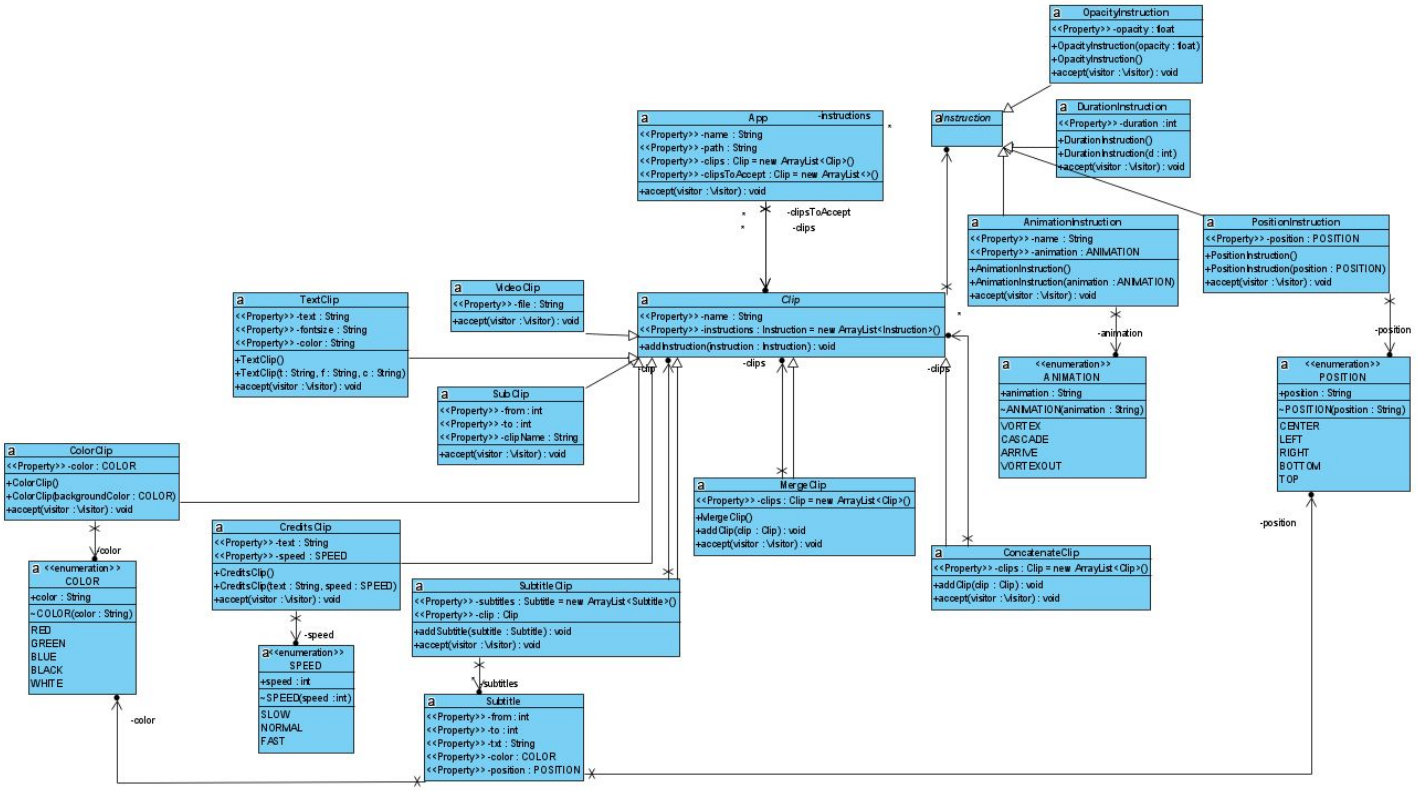
Le but du projet était de développer un langage permettant à un néophyte du développement de faire du montage vidéo. Pour ce faire, nous utilisons la librairie MoviePy.

Nous utilisons groovy pour traduire le langage en python et ainsi utiliser la librairie. De plus nous proposons une interface graphique en Java qui permet à l'utilisateur d'avoir de la coloration syntaxique et de l'autocomplétion tout en pouvant traduire dans la même fenêtre et sans commande le langage en python et de pouvoir run le code python pour monter sa vidéo. Le code se trouve sur ce le lien suivant :

https://github.com/LoicBertin/Circular-DSL-CinEditorML

I - Description des langages développés

I.1) Schéma du domain-model



Notre domain model (également visible dans notre github dans le dossier "rendu") est basé sur deux principes clefs: les clips et les instructions.
Il existe de multiples types de clips (définis en fonction des types de clips de MoviePy) qui possèdent une liste d'instructions. Ces instructions permettent de rajouter une liste de spécificités communes aux clips mais non obligatoire à leur bon fonctionnement. De ce fait, l'instruction de durée ne devrait pas exister car il s'agit d'une valeur obligatoire au bon fonctionnement des clips de MoviePy (ce que nous ne savions pas lors de la mise au point de ce modèle) mais nous n'avons pas eu le temps de refactorer ce problème de conception. En ce qui concerne les textes dans nos vidéos, il y a une spécificité à notre modèle. En effet, MoviePy propose des TextClip qui sont juste des champs de texte mais qui ne fonctionnent

pas s'ils ne sont pas placés sur un fond (coloré ou une vidéo). Ces textes sont utiles mais ne permettent pas de placer des textes en plein milieu d'un autre clip à moins de superposer de nombreux clips (nous en parlerons plus tard dans la partie III.1). Pour régler ce problème nous avons donc utilisé un autre type de clip fourni par MoviePy qui est un SubtitleClip qui se rapproche d'une liste de TextClip avec des temporalités ajoutées. De ce fait les TextClip sont voués à disparaître dans notre implémentation au profit des SubtitleClip qui offrent plus de libertés.

I.2) Syntaxe du langage

<importVideoClip> ::= “importVideoClip” <path (to the file)> “named” <name (given to the clip)>

<subClipOf> ::= “subClipOf” <name (of the clip)> “from” <beginningTime> “to” <endTime> “named “name (given to the clip)>

<makeVideoClip> ::= “makeVideoClip” <name (of the clip)> “with” <name (of the clip to add)> (“then” <name (of the video clip to add)>)*

<export> ::= “export” <name (of the clip to export)> “at” <path (to the file exported)>

<createClip> ::= “createClip” <name (of the clip)> (“during” <during>)? “with_background” <background> “with_text” <content (to write on the clip)> “at” <POSITION> “from” <beginningTime> “to” <endTime> (“and_with_text” <content (to write on the clip)> “at” <POSITION> “from” <beginningTime> “to” <endTime>)*

<addText> ::= “addText” <content (to write on the clip)> “on” <name (of the clip to apply text)> ((“backward”|”during” <time (before the end or after the start)>) | “from” <beginningTime> “to” <endTime>) (“and_on” <name (of the clip to apply text)> (“backward”|”during” <time (before the end of after the start)> | “from” <beginningTime> “to” <endTime>))*)?

<path> ::= (<letter>+ | “/” | “\”)+

<name> ::= <letter>+

<content> ::= <letter>+

<letter> ::= "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z" | "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"

<digit> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

<duration> ::= <digit>+

<beginningTime> ::= <digit>+

<endTime> ::= <digit>+

<time> ::= <digit>+

<POSITION> ::= “CENTER” | “TOP” | “BOTTOM” | “LEFT” | “RIGHT”

(background est une couleur si jamais il y a le mot clef during sinon c’est un nom de clip

<background> ::= “during” ? <color> : <name>

<color> ::= “RED” | “GREEN” | “BLUE” | “BLACK” | “WHITE”

<makeCreditsWith> ::= “makeCreditsWith” <content (all the names separated with a comma)> “named” <name (given to the credit clip)> “at_speed” <SPEED>

<SPEED> ::= “SLOW” | “NORMAL” | “FAST”

<text> ::= “text” <content (to write on the clip)> “named” <name (given to the text clip)> “during” <duration> “at” <POSITION> “animated_with” <ANIMATION>

<ANIMATION> ::= “VORTEX” | “CASCADE” | “VORTEXOUT” | “ARRIVE”

DEPRECATED (those below are deprecated even if they're working because they are not so easy to use for someone which have never developed and the version written before is easier to understand):

<backgroundClip> ::= "backgroundClip" <color> "named" <name (given to the color clip)> "during" <duration>

I.3) Description de notre extension et de comment nous l'avons développée

Nous avons choisi l' extension "Support for customized animated text" et nous avons implémenté les deux scénarios.

Le premier est un générique de fin, comme nous pouvons le voir ci-dessus on le crée avec la méthode "createCreditsWith", on écrit ensuite toute les personnes que nous voulons citer séparés par une virgule et enfin on précise la vitesse à laquelle le texte doit défiler (slow, normal ou fast). Afin d' être sûr de voir tous les noms lors du générique, un simple calcul a été fait (0,8 seconde par collaborateur en fast, 1,2 en normal et 2,4 en slow), qui en fonction de la vitesse choisie et du nombre de participants, calcul la durée du clip.

Le deuxième scénario est un texte animé. C'est une caractéristique qui vient se rajouter au TextClip avec le mot clé <text> à la fin de celui-ci avec le mot-clé "animated_with". Il y a 4 animations implémentées : le vortex, la cascade, le vortexout et une arrivée par la gauche.

I.4) Description de l'implémentation du compilateur java

Nous avons repris le compilateur fourni pour ArduinoML que nous avons retravaillé pour correspondre à ce nouveau projet. Nous avons donc toujours le système de "visitor" et de "accept" afin de réaliser la conversion d'objet java vers code python. Grâce à ce système nous pouvons obtenir du code python facilement modulable à notre guise. Ce code python est écrit sur la sortie standard, il est donc facilement récupérable pour notre "front" en java.

Nous avons ajouté au tout début du "toWiring" 8 fonctions python qui seront ensuite utilisées pour optimiser la génération des vidéos ou ajouter des effets sur les textes.

II - Scénarios implémentés

Nous avons implémenté les deux scénarios basiques consistant :

- **1er scénario :** Concaténation d'un fond noir de 10 secondes avec écrit au milieu quelque chose suivi de deux vidéos puis d'un fond noir de 15 secondes avec écrit au milieu autre chose
- **2ème scénario :** Concaténation d'un fond noir de 10 secondes avec écrit au milieu quelque chose suivi d'une vidéo étant un clip d'un autre vidéo entre deux intervalles sur laquelle on a ajouté du texte durant les 10 premières secondes puis à 40 secondes du clip un second texte et enfin sur les 5 dernières secondes un dernier texte. Ce texte sera d'ailleurs toujours affiché durant les 10 secondes du clip vidéo suivant qui est une partie d'une vidéo (dans notre cas la même vidéo) et enfin un dernier fond noir de 10 secondes avec un message écrit en son centre.
- **1er et 2ème Scénario d'extension :** Concaténation d'un fond noir avec un texte animé avec l'animation cascade, une vidéo classique puis un générique de fin qui scroll avec la vitesse fast
- **Scénario de démonstration du DSL :** Concaténation d'un fond noir durant 5 seconde avec un texte au centre; avec une vidéo importée qui a été coupé à 23 et 43

secondes avec l'ajout de deux textes à des timestamp différent; avec une seconde vidéo importé qui a un texte sur les deux vidéo importés; avec les crédits.

III - Analyse de notre implémentation

III.1) CinEditorML

Notre implémentation permet de mettre des fonds colorés ou des vidéos d'une durée souhaitée avec du texte écrit à différentes positions. Les vidéos sur lesquelles sont appliqués les textes peuvent être découpées en subclips. Dans le but de placer des textes à une temporalité particulière de la vidéo, nous avions dans un premier temps ajouté sur le clip en question un filtre par dessus qui contenait du début du clip jusqu'au moment où le texte devait apparaitre un texte vide suivi du texte en question et ce filtre était ensuite fusionné avec le clip. Toutes ces opérations rendent le montage de la vidéo très long (une dizaine de minutes pour deux minutes de vidéo). On a donc cherché une autre solution, celle-ci consiste à utiliser les SubtitlesClip qui sont fournis par MoviePy pour décrire l'intégralité des textes de la vidéo (donc quand il ne doit pas y avoir de texte on doit mettre un texte vide) puis on concatène directement sur le clip les textes les uns à la suite des autres. Pour que l'utilisateur de notre langage n'ait pas à rajouter des textes vides quand il n'y a pas de texte, nous analysons les temporalités des subtitles pour remplir les vides avec des textes vides. Toutefois, nous n'avons pas eu le temps de réordonner les textes dans un SubtitlesClip pour que ceux-ci soient dans un ordre chronologique même si l'utilisateur ne le renseigne pas dans l'ordre chronologique pour la vidéo. Cela peut-être gênant mais pas si compliqué à corriger, il suffirait de trier les textes pour qu'ils soient replacés dans l'ordre chronologique à la transformation de groovy à python. Nous pouvons donc donner une durée à chacun de nos clips ainsi que pour les textes une position sur l'écran et il serait très simple de changer la couleur il faudrait rajouter un mot clef sur les phrases du langage car actuellement nous mettons le texte en blanc pour chacun des textes créés.

Pour l'ajout de texte sur 2 clips nous avons mis en place une grammaire permettant de faire des actions plus poussées sur les clips mais rendant sa compréhension moins aisée.

Lorsque la commande addText est utilisée elle peut permettre d'ajouter du texte sur "n" clips déjà existants à leur début ou à leur fin. Cela permet 3 usages différents :

- ajouter du texte au début de plusieurs clips, comme un sous titre commun à tous les clips
- ajouter du texte à la fin de plusieurs clips
- ajouter du texte à la fin d'un clip et au début d'un autre, avoir donc du texte sur 2 clips comme demandé dans le scénario 2.

Cette option d'ajout de 5 secondes avant la fin d'un clip, par exemple, est assez aisé en python (puisqu'il suffit de récupérer la durée du clip et d'enlever 5), mais devient plus compliqué lorsqu'il faut récupérer et passer l'information du DSL groovy au kernel java. Pour se faire nous avons ajouter un mot clé dans notre langage "before_the_end", lorsque ce mot clé est lu notre groovy crée un subtitle clip qui se termine à 9999. Lorsque le kernel lit 9999 il sait que c'est un sous titre qui va être avant la fin d'un clip et le transforme donc correctement en code Python. Cette petite partie de code sioux aurait pu être corrigé de 2 façons :

- rendre SubtitleClip abstrait et faire 2 classes filles : SubtitleClipAtStart et SubtitleClipBeforeTheEnd
- ajouter dans SubtitleClip un attribut subtitleType qui aurait pris la valeur afterStart ou beforeTheEnd

Pour l'extension, les 2 scénarios ont un comportement assez différents, l'implémentation n'est donc pas la même. Pour le texte animé, une nouvelle instruction au TextClip a été ajoutée "animated_with" <ANIMATION> et les différentes animations que j'ai inclus dans le projet sont dans une énumération. Cette implémentation a donc pour limite les animations que j'ai inclus dans le langage, si nous voulons de nouvelles animations, il faudra qu'un développeur les ajoute au kernel.

Pour le générique de fin, un attribut sur un texte clip n'était pas suffisant selon moi, et un nouveau type de clip a été ajouté (CreditsClip) et donc avec sa nouvelle syntaxe BNF qu'on peut voir ci-dessus (makeCreditsWith). L'utilisateur peut entrer le nombre de noms qui lui convient, le temps de défilement est automatiquement géré par notre langage. La vitesse est "normal" par défaut, mais peut être modifiée à l'aide de l'énumération SPEED que nous avons mise en place. L'utilisateur est donc limité à 3 choix, slow, normal et fast. Nous aurions pu mettre en place un système d'échelle de 1 à 10 pour gérer la vitesse, mais je trouvais cela plus compréhensible pour l'utilisateur final de proposer des mots clairs.

Il aurait été intéressant de rajouter diverses options comme par exemple la couleur du texte (il suffirait de rajouter un mot au vocabulaire pour set l'attribut de couleur du texte, qui existe déjà dans le kernel). Cette option n'étant pas présente, pour la vidéo de démonstration nous avons changé dans le ToWiring pour que le texte par défaut soit écrit en rouge et soit lisible (alors qu'il est par défaut en blanc) mais nous n'avons pas laissé ce changement pour que le texte des autres vidéos reste par défaut blanc (ce qui est plus classique). Donc si vous souhaitez reconstruire la vidéo de présentation les textes écrits seront en blanc (sur fond majoritairement blanc..). Il aurait pu être intéressant aussi de laisser l'utilisateur pouvoir renseigner le nombre de thread qu'il souhaite utiliser lors de la création de la vidéo pour accélérer l'exécution de MoviePy (cela doit rester une option pour qu'un utilisateur néophyte n'ai pas à savoir ce qu'est un thread et combien de thread il peut utiliser sans que sa machine tombe).

III.2) Technologies utilisées

Nous avons utilisé Groovy pour l'implémentation du DSL. Ce DSL interne avait déjà été utilisé par plusieurs membres de l'équipe pour le premier projet. Nous l'avons également choisi car le fonctionnement par classe de java rend la collaboration plus aisée que MPS.

De plus, nous avons tous des connaissances en Java contrairement à MPS où seulement 2 membres de l'équipe avait développé du code pour le premier projet. Pour finir cela nous permet d'avoir un projet centré autour d'un seul langage et de rester dans un environnement Java avec notamment l'application qui est développée à l'aide de la librairie graphique Swing.

IV - Application

L'application réalisée en java permet d'avoir une interface "intuitive", afin d'utiliser CinEditorML. Comme précisée plus haut, l'application est totalement indépendante du DSL et ne nécessite que le .jar du DSL pour fonctionner ainsi que Python installé avec les bonnes librairies.

La capture d'écran située plus bas permet de visualiser l'interface de l'application.

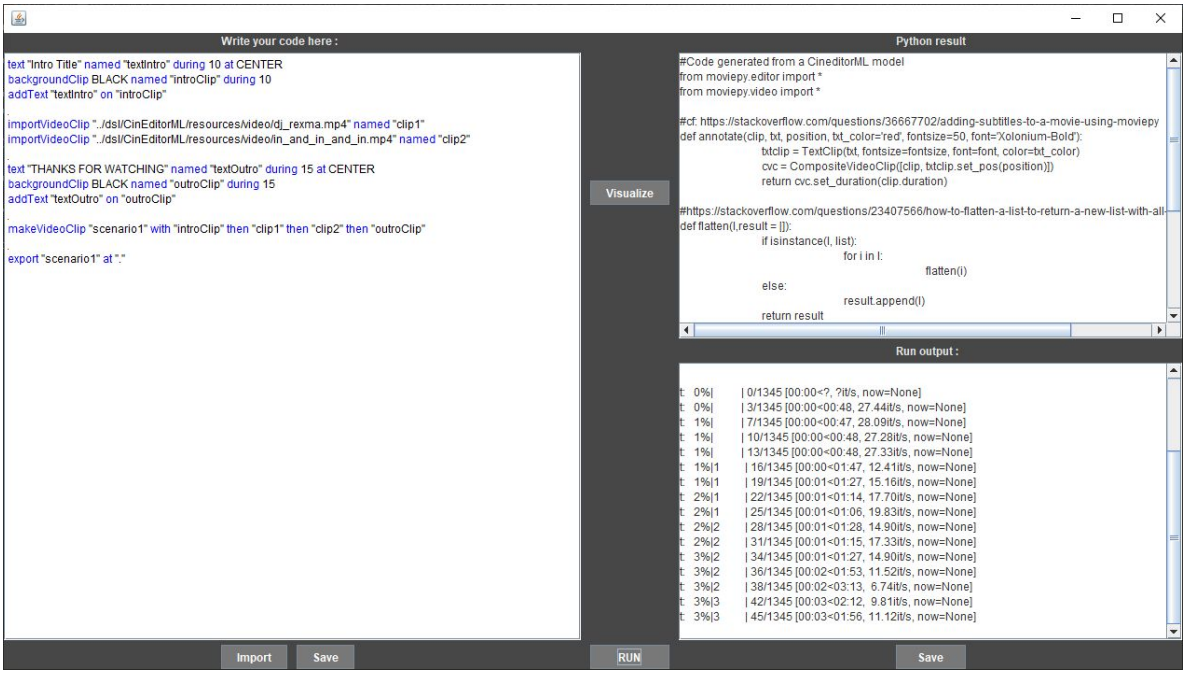
La fenêtre de gauche permet d'écrire son code dans le langage de notre DSL et dispose d'une autocomplétion (pas forcément fonctionnelle à 100% mais qui marchera dans la

plupart des cas) ainsi que du highlight de code (en bleu les mots clés) avec de la détection d'erreur basique.

Les 2 boutons en dessous de cette fenêtre permettent d'importer du code depuis un fichier ou de sauvegarder le code écrit dans un fichier. Dans le cas de la sauvegarde, l'extension du fichier est ajoutée automatiquement.

La fenêtre en haut à droite permet, après avoir cliqué sur "Visualize" de prévisualiser son code en python. Le bouton save tout en bas permet de sauvegarder son code python (l'extension .py est ajoutée automatiquement).

Enfin, lorsque l'on appuie sur le bouton "RUN", cela exécute le code python et le résultat de l'exécution est écrit dans la fenêtre en bas à droite. Il faut bien attendre la seconde popup pour que le traitement de la vidéo soit complètement fini.



V - Séparation du travail

Guillaume : réalisation de l'application Java

Loïc : amorçage du domain model puis amorce application java puis réalisation du DSL pour finir le scénario 2

Stéphane : écriture du DSL pour scénario 1 et 2

Virgile : réalisation des 2 scénarios d'extension.

VI - Vidéo de présentation du langage

https://github.com/LoicBertin/Circular-DSL-CinEditorML/blob/main/rendu/videoPresentationLangage.webm

Cette vidéo a été réalisé à l'aide de notre DSL, son script est disponible ici : https://github.com/LoicBertin/Circular-DSL-CinEditorML/blob/main/dsl/CinEditorML/resources/groovy/ScenarioMontagePrez.groovy