06/03/2021

SI5 Domain Specific Languages
Report Project 2
Internal DSL CinEditorML

Our team

Lydia BARAUKOVA (SI5 AL)
Jean-Marie DORMOY (SI5 WEB-IA)
Gabriel REVELLI (SI5 IHM)

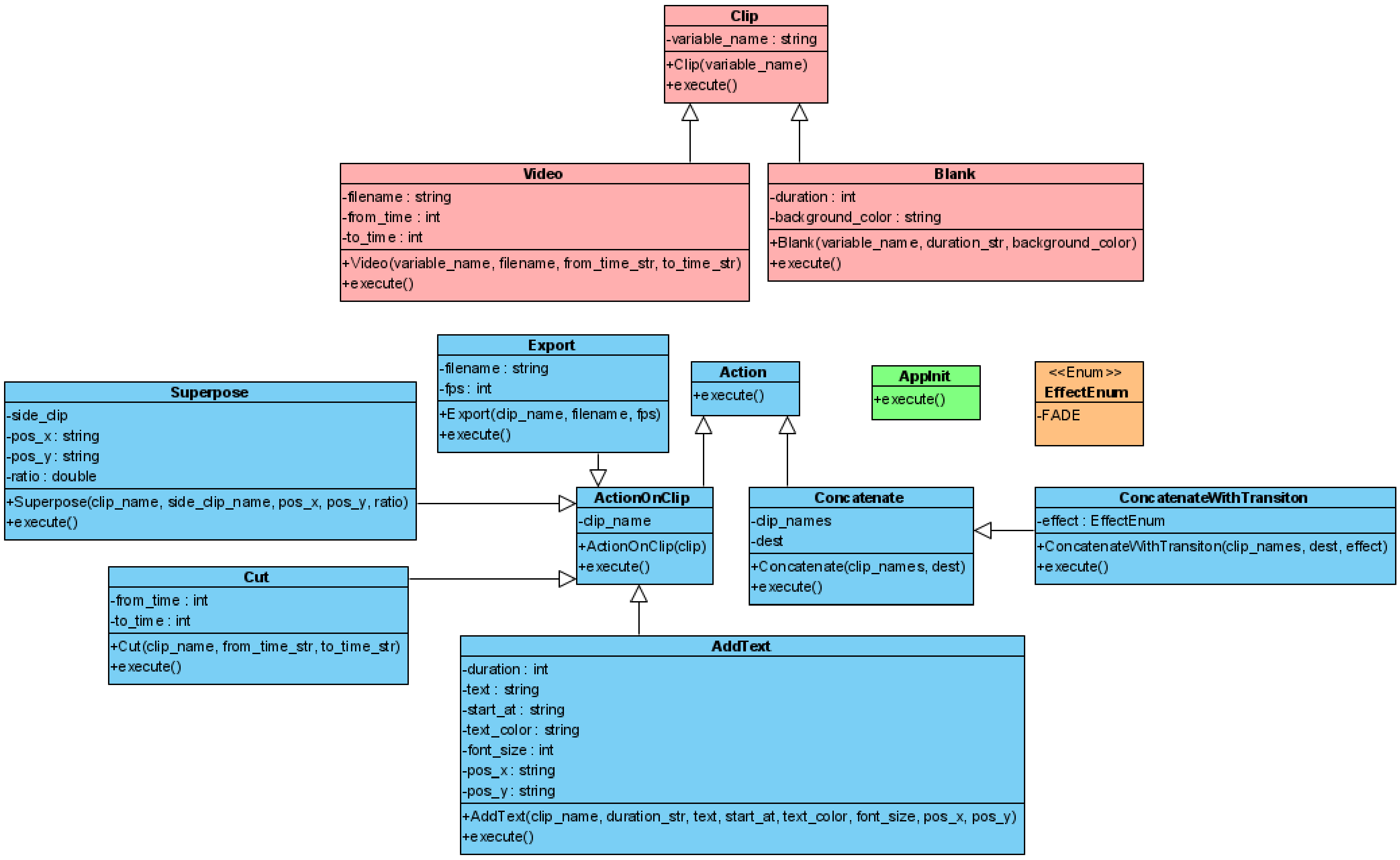
Link to our code

https://github.com/Livelinndy/si5-dsl-team-a/tree/main/Project2-CinEditorML

Description of the developed DSL

Domain model

The domain model contains representations of different possible actions. For example, *Video* corresponds to initialization of a variable containing a video from a file; *Blank* corresponds to initialization of an empty clip containing only a colored background; *AddText* corresponds to adding text over a clip; concatenate corresponds to concatenation of multiple videos; etc. Each class has a method *execute()* to produce code relevant to the action.



Concrete syntax

Concrete syntax represented in a BNF-like form:

```
<root>              ::= <actions>

<actions>           ::= ( <init> | <add-sth> | <concat> | <export> )*

<init>              ::= "clip" <IDENTIFIER> ( <video> | <blank> )

<video>             ::= "is" <MP4> <subclip>?
<subclip>           ::= "from" <TIME> "to" <TIME>

<blank>             ::= <background> <duration>

<background>        ::= <color> "background"
<color>             ::= "color" <HEXCOLOR>

<duration>          ::= "during" <TIME>

<add-sth>           ::= "add" ( <add-text> | <add-clip-on-top> )

<add-text>          ::= <text-type> <TEXT> ( "at" ( <TIME> | "end" ) <duration> )?
<text-type>         ::= ( "title" | "subtitle" | ("text" <POSITION> <NUMBER>? ) )

<add-clip-on-top>   ::= "clip" "on" <POSITION> "scale" <RATIO> "to" <IDENTIFIER>

<concat>            ::= "concat" <IDENTIFIER> "and" <IDENTIFIER> <concat-additional>* <with-transition>? "to" <IDENTIFIER>
<concat-additional> ::= "and" <IDENTIFIER>

<with-transition>   ::= "with" "transition" <transition-type>
<transition-type>   ::= "FADE"

<export>            ::= "export" <IDENTIFIER> "as" <MP4>

<MP4>               ::= """ <IDENTIFIER> ".mp4""

<POSITION>          ::= <POS_X>? <POS_Y>?
<POS_X>             ::= "left" | "center" | "right"
<POS_Y>             ::= "top" | "center" | "bottom"

<HEXCOLOR>          ::= "#" [0-9abcdef]{3,6}

<TIME>              ::= "-"? ( ( "\d{1,2}h") ? "\d{1,2}m" )? "\d{1,2}s"
<TEXT>              ::= """ ( <IDENTIFIER> | <WHITESPACE> )+ """
<RATIO>             ::= ( "0" | "1" ) "." <DIGIT>

<NUMBER>            ::= <DIGIT>+
<IDENTIFIER>        ::= ( <LOWERCASE> | <UPPERCASE> ) ( <LOWERCASE> | <UPPERCASE> | <DIGIT> )+
<LOWERCASE>         ::= [a-z]
<UPPERCASE>         ::= [A-Z]
<DIGIT>             ::= [0-9]

<NEWLINE>           ::= ("\r"? "\n" | "\r")+        -> skip
<WHITESPACE>        ::= ((" " | "\t")+)         -> skip
<COMMENT>           ::= "//" ~( "\r" | "\n" )*      -> skip
```

Implementation

The language we implemented is an external DSL. We made it using Python and the moviepy (https://github.com/Zulko/moviepy) library.

First, we wanted to implement an embedded DSL in Python but we found some difficulties while doing it. We didn't have a good example of an embedded DSL with Python without method chaining. We developed the DSL kind of in our own way and ended up with an external DSL.

On one hand we have the kernel package. It contains the domain model classes. As we previously mentioned, each class corresponds to an action. The *execute()* method of each action generates code relevant to this action. So the kernel is responsible only for code generation.

On the other hand we have the src package containing our parser. It is responsible for reading CinEditorML code from a file, linking it with the classes of the domain model, invoking code generation and executing the obtained code.

Compiler

Our parser reads our CinEditorML code line by line. For each line it identifies the associated action and instanciates it with the provided parameters and stores it in a list of actions.

```
actions = [AppInit(),
           # (a) add a title on a black background at the beginning for 10 seconds
           Blank('t1', '10s'),
           AddText('t1', '10s', 'title: frogs'),
           # (b) add a first video clip that appears directly after the title screen
           Video('c1', 'frogs.mp4'),
           # (c) add another video clip that appears directly after the first clip
           Video('c2', 'frogs2.mp4'),
           # (d) add a thanks sentence at the end, lasting for 15 seconds
           AddText('c2', '15s', 'thanks for watching', start_at = '1m55s'),
           # (e) export the result as a video file
           Concatenate(['t1', 'c1', 'c2'], 'cf'),
           Export('cf', 's1_kernel.mp4')]
```

Example of an actions list

Then all the sequence of actions is executed (*execute()* method of each action is called) to obtain Python code, it is concatenated into a single string.

```
code = ""
for a in actions:
        code += a.execute()
```

Concatenation of the generated code

```
import os
import sys
sys.path.append('../')
import moviepy.editor as mp
t1 = mp.ColorClip(size = (256, 144), color = [0, 0, 0], duration = 10)
tc = mp.TextClip('title: frogs', fontsize = 20, stroke_color = 'red', stroke_width = 1.5)
tc = tc.set_pos(('center', 'center')).set_duration(10)
t1 = mp.CompositeVideoClip([t1, tc])
c1 = mp.VideoFileClip(os.path.join('../resources/videos', 'frogs.mp4')).resize(newsize = (256, 144))
c2 = mp.VideoFileClip(os.path.join('../resources/videos', 'frogs2.mp4')).resize(newsize = (256, 144))
tc = mp.TextClip('thanks for watching', fontsize = 20, stroke_color = 'red', stroke_width = 1.5)
tc = tc.set_pos(('center', 'center')).set_duration(15)
tc = tc.set_start(t = 115)
c2 = mp.CompositeVideoClip([c2, tc])
cf = mp.concatenate_videoclips([t1,c1,c2])
cf.write_videofile(os.path.join('../output', 's1_kernel.mp4'), 30)
```

Example of generated code

Finally this Python code is executed using Python's *exec(str)* method.

Additions

As an addition for our DSL project, we have created a Notepad++ syntax colorizer. It will help users to distinguish different keywords of our DSL and make the CinEditorML code more readable.

You can find our syntax colorizer here: https://github.com/Livelinndy/si5-dsl-team-a/tree/main/Project2-CinEditorML/NppSyntaxColorizer

Scenarios

Basic scenarios

1. **Scenario 1: support for video concatenation, title screens with choice of background color, adding titles over video and video export**
    a) add a title on a black background at the beginning for 10 seconds.
    b) add a first video clip that appears directly after the title screen.
    c) add another video clip that appears directly after the first clip.
    d) add a thanks sentence at the end, lasting for 15 seconds.
    e) export the result as a video file.

    This scenario written in our DSL is:

```
clip t1 color #000 background during 10s
add title "title: frogs" to t1 during 10s
clip c1 is "frogs.mp4"
clip c2 is "frogs2.mp4"
add title "thanks for watching" to c2 at 1m55s during 15s
concat t1 and c1 and c2 to cf
export cf as "s1_res.mp4"
```

2. **Scenario 2: support for video cutting and subtitles**
    a) add a title on a black background at the beginning for 10 seconds.
    b) load a first video clip *clip1*.
    c) create two clips *clip1a* and *clip1b* respectively taken from 01:33 to 01:46 min and from 02:03 to 02:10 min.
    d) add a subtitle to *clip1a* from the beginning and for 10 seconds, followed by another subtitle starting 30 seconds after the end of the first one and visible for 10 seconds.
    e) add a subtitle to *clip1b* starting 5 seconds before *clip1b* and lasting for 15 seconds.
    f) add a thanks conclusive text on a fixed background color (let's say black).
    g) export the result as a video file.

    This scenario written in our DSL is:

```
clip t1 color #000 background during 10s
add title "introduction title" to t1 during 10s
clip c1a is "frogs.mp4" from 9s to 1m08s
clip c1b is "frogs.mp4" from 1m49s to 2m10s
add subtitle "subtitle s1a" to c1a during 10s
add subtitle "subtitle s2a" color #f0f to c1a at 40s during 10s
add subtitle "subtitle s1b" to c1b during 15s
clip t2 color #000 background during 10s
add title "thanks for watching" to t2
concat t1 and c1a and c1b and t2 to cf
export cf as "s2_res.mp4"
```

Extension scenarios

3. **Scenario 3: support for stacking and transitions between clips**
   By implementing this extension it becomes possible to add transitions between the various clips that are concatenated. It must also be possible to stack various clips one onto the other with specific sizes.
   a) add fade out fade in transitions when creating the main video.
   b) stack a side video over the main video in the corner.
   c) export the result as a video file.

P.S. Of course the fade effect does not affect the side video, which stays in the corner.

This scenario written in our DSL is:

```
clip c1side is "frogs.mp4" from 33s to 1m05s
clip c1a is "frogs.mp4" from 9s to 22s
clip c1b is "frogs.mp4" from 1m33s to 1m45s
clip c1c is "frogs.mp4" from 2m03s to 2m10s
concat c1a and c1b and c1c with transition FADE to cf
add clip c1side on bottom right scale 0.3 to cf
export cf as "s3_res.mp4"
```

Critical analysis

1) Of the DSL implementation with respect to CinEditorML use cases:

Our implementation allows us to put text over videos and to change the text position and color. This way we didn't have to implement 2 separate classes AddTitle and AddSubtitle, we have just AddText that accepts the necessary parameters from our parser.

Our implementation doesn't allow you to add text before the start of a clip or relatively to the end of the clip. We can only add it relatively to the beginning of the clip. This is due to the fact that in the kernel we operate only attributes to produce string lines of code, we don't have access to the moviepy video objects to get their duration to calculate time relative to the end of the video.

We implemented only one type of transition between videos - FADE. We could have implemented some more transitions to have a choice (for example, slide, blur, etc.), but we didn't do it because of the lack of time available for this project and since it wasn't necessary to demonstrate our use cases.

2) Of the technology we chose to achieve it:

Python is a widespread programming language. Python 3 is preinstalled on many Debian Linux operating systems. And if Python 3 is not installed, it's very easy to install by going to the Python official website (https://www.python.org/downloads/) and downloading an executable, or by running a command in a terminal (in case of a linux-based system).

The moviepy library used in the kernel is easy to install with pip (distributed along with Python 3) by running the command *‘pip install moviepy’*.

To work properly, moviepy also needs imagemagick which has to be downloaded and installed separately (https://imagemagick.org/script/download.php). Making it work correctly could be a bit tricky, but the explanations on the imagemagick website are very clear, so by following these instructions even an inexperienced user can do the setup.

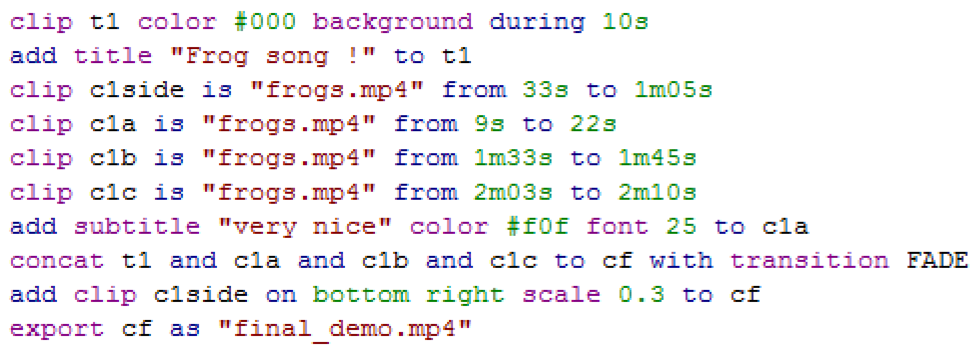
Usage example

You can find a demo video of our DSL here:
https://github.com/Livelinndy/si5-dsl-team-a/tree/main/Project2-CinEditorML/pyCinEditorML/output

The *final_demo.mp4* video shows all the features of our DSL that we previously mentioned in our scenarios:

- Loading videos
- Cutting videos on loading
- Creating empty colored screens
- Adding text with different parameters over videos or empty screens
- Concatenating videos
- Concatenating videos with transitions
- Stacking videos
- Exporting videos

Here’s our demo scenario:



```
clip t1 color #000 background during 10s
add title "Frog song !" to t1
clip c1side is "frogs.mp4" from 33s to 1m05s
clip c1a is "frogs.mp4" from 9s to 22s
clip c1b is "frogs.mp4" from 1m33s to 1m45s
clip c1c is "frogs.mp4" from 2m03s to 2m10s
add subtitle "very nice" color #f0f font 25 to c1a
concat t1 and c1a and c1b and c1c to cf with transition FADE
add clip c1side on bottom right scale 0.3 to cf
export cf as "final_demo.mp4"
```

For our demos we use some low quality videos (256x144 px) in order to avoid waiting too much time for the processing to finish.

You can also run each of our scenarios by going to the *src* folder and running the command “*Main.py X*” where X is the name of the scenario (s1, s2, s3 or demo). The resulting video will be in the *output* folder.

P.S. The *Main.py* file is a runnable Python script that accepts scenario name as a parameter.

Responsibilities of each team member

Lydia BARAUKOVA	concrete and abstract syntax definition, writing of working code snippets using moviepy, code setup, kernel implementation, assembling parser results with kernel objects, report redaction
Jean-Marie DORMOY	concrete and abstract syntax definition, starting kernel implementation, parser implementation with regexes
Gabriel REVELLI	concrete and abstract syntax definition, creation of a Notepad++ syntax colorizer for our DSL, starting parser implementation, report redaction, creation of the demo video