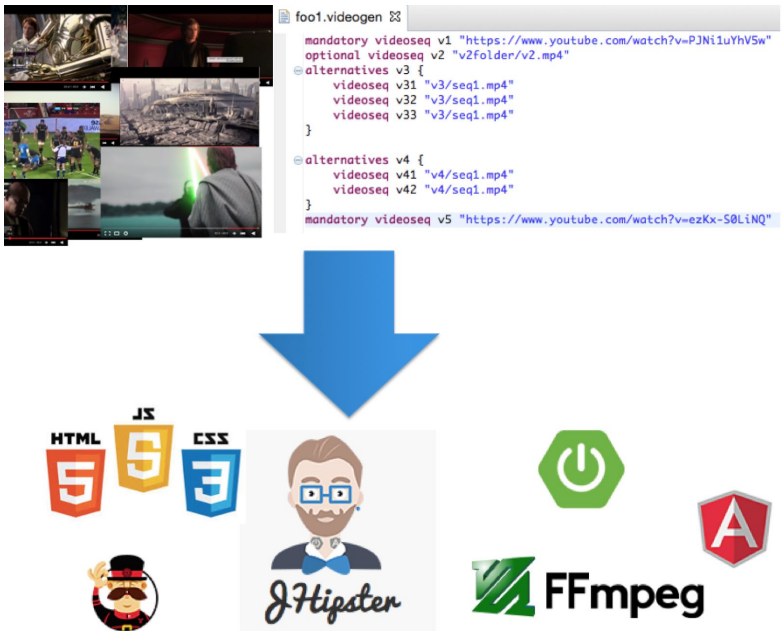
Ingénierie des Modèles

TP3, TP4, TP5, TP6 –
Métamodélisation, DSL, Transformations (2015)

L'objectif de ce projet est de déployer un générateur ou configurateur Web de vidéos à partir d'une spécification textuelle (VideoGen). Les utilisateurs du site Web pourront par exemple visualiser des variantes de vidéos générées aléatoirement ou avec des probabilités.
Le projet peut être vu comme une généralisation du générateur de "Bref 30 ans" (http://bref30ans.canalplus.fr/). Outre la généralisation de cette idée, nous incorporerons d'autres fonctionnalités configurables à notre générateur de générateur.



1. Donner vie à VideoGen

Objectifs: maîtrise de la transformation de modèles (model-to-text); tests; Web; TP3

De manière générale on souhaite maintenant appliquer un ensemble de transformations (model-to-model, model-2-text) pour refactoriser, vérifier la cohérence, ou obtenir de nouveaux artefacts (e.g. Web) à partir d'une spécification VideoGen.
Les spécifications (modèles) ne seront plus des modèles contemplatifs pour communiquer ou documenter un système ; on les manipulera comme n'importe quel programme.
Dans la suite on s'appuiera sur la grammaire Xtext que vous avez développée au TP2, i.e., les spécifications VideoGen seront conformes à votre grammaire.

Comme premier objectif **essayons de générer concrètement des variantes de vidéo à partir de spécifications VideoGen**. Pour y parvenir, nous allons écrire une transformation (model-to-text) qui prend en entrée n'importe quelle spécification VideoGen et génère en sortie une liste aléatoire de séquences vidéo[1].
Cette liste sera ensuite exploitée par ffmpeg (https://ffmpeg.org) pour assembler les différentes séquences vidéo en une seule vidéo. On s'appuiera par exemple sur ce document pour assembler ("concaténer") les vidéos: https://trac.ffmpeg.org/wiki/Concatenate

```
# this is a comment
file 'v3/seq1.avi'
file '/path/to/video2.avi'
file '/path/to/video3.avi'
```

FFmpeg

La liste (ou "playlist") est générée aléatoirement de la façon suivante : chaque séquence vidéo obligatoire est évidemment incluse ; pour les séquences optionelles, on l'incorpore dans 50% des cas ; on effectue enfin un tirage équiprobable pour sélectionner une et une seule séquence vidéo alternative.

Q1: Créer un nouveau projet Xtend et écrire la transformation. Tester votre solution sur un jeu de données[2]. Pour l'instant on se contentera d'une solution qui génère, par appel du code Xtend, la liste ffmpeg; on pourra appeler ffmpeg dans un 2ème temps pour obtenir la vidéo (vous pouvez également intégrer la procédure d'appel à ffmpeg dans Xtend/Java mais ne perdez pas trop de temps sur cet aspect là du travail).

2. Une autre vie pour VideoGen (Un métamodèle intermédiaire)

Objectifs: maîtrise de la transformation de modèles (model-to-model); métamodélisation; TP3

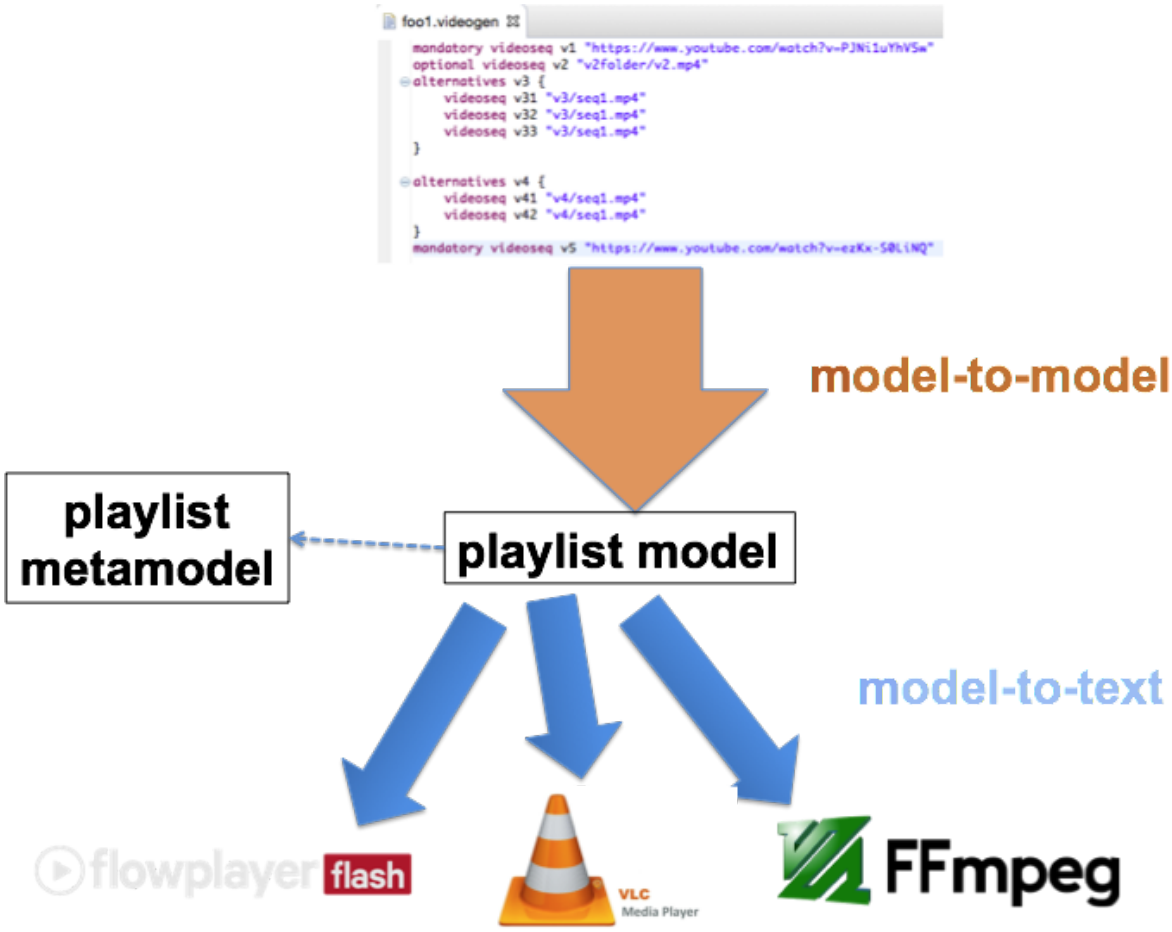
[1] Pour le choix des vidéos, vous êtes libre… soyez créatif! Attention cependant à ne pas divulguer certaines vidéos publiquement (e.g., dans un dépôt github public).
[2] Il est conseillé de considérer des vidéos dans des formats classiques, homogènes (e.g., mp4) pour avoir une première démonstration assez rapide. Par ailleurs "jeu de données" veut aussi dire un ensemble de spécifications VideoGen (votre procédure doit marcher pour n'importe quelle spécification VideoGen "valide")

Plutôt que d'utiliser ffmpeg pour assembler les séquences vidéo et produire une nouvelle vidéo complète, on va lire les séquences une à une avec un lecteur vidéo (e.g., VLC ou un lecteur Web) et une "playlist".
La solution ffmpeg est en effet coûteuse en temps et en espace mémoire, et de fait se prête assez mal à un usage Web. Le fait de lire en streaming les séquences vidéo présente des avantages lors d'une mise en ligne de la solution. A noter que la solution ffmpeg a aussi des qualités et pourra s'intégrer avec le lecteur vidéo Web – on peut par exemple envisager de proposer l'export et la sauvegarde de la variante une fois la vidéo visualisée avec le lecteur Web.

Le défi maintenant est de générer une playlist qui sera ensuite jouée par un lecteur. On considère encore un usage local pour jouer les variantes de vidéo: on utilisera VLC pour lire la playlist.

Le principe est le même que pour la première question: il s'agit de générer une "playlist", mais dans un format textuel différent, compréhensible par un lecteur vidéo. C'est le format M3U qui est considéré ici: https://en.wikipedia.org/wiki/M3U (en première approche on considérera le format simple, non étendu de M3U).

On pourrait tout à fait envisager une transformation model-to-text, mais on va utiliser une approche différente, résumée dans la figure ci-dessous.
Une spécification VideoGen (modèle) est d'abord transformée en un autre modèle, conforme à un métamodèle de playlist. L'idée est que ce modèle est ensuite transformé en une représentation textuelle concrète (liste ffmpeg, M3U, M3U étendue, etc.)



Q2: Créer un nouveau projet et produire un métamodèle de "playlist" tel que les constructions de ce métamodèle permettent d'exprimer des instances de liste ffmpeg et de playlist .m3u. Autrement dit, à partir d'une instance du métamodèle de playlist (un modèle), on doit être capable de générer une représentation textuelle de la liste ffmpeg et de la playlist .m3u compréhensible par VLC.

Q3 : Ecrire une transformation model-to-model qui prend en entrée une spécification VideoGen et qui produit en sortie une instance de playlist (un modèle conforme au métamodèle de playlist). Ecrire ensuite une transformation model-to-text qui prend en entrée un modèle de playlist et qui produit en sortie un fichier texte .m3u compréhensible par VLC.

Q4 : Ecrire une transformation model-to-text qui prend en entrée un modèle de playlist et qui produit en sortie une "liste" compréhensible par ffmpeg (comme dans Q1).

Nous allons maintenant générer un autre format de playlist, compréhensible par le lecteur Flowplayer. Une contrainte est cependant donnée[3]: le format “M3U extended” doit être utilisé (cf ci-dessous) et un bout de code vous est fourni en Annexe pour opérationaliser la lecture avec la playlist.

Q5 : Réfléchir à une transformation model-to-text qui prend en entrée un modèle conforme au métamodèle de playlist et qui produit en sortie un fichier texte au format “M3U étendu”, compréhensible par le lecteur Flowplayer (cf ci-dessous). Quelles sont vos recommandations pour y parvenir[4]?

```
#EXTM3U
#EXT-X-DISCONTINUITY
#EXTINF:3
resources/videos/vp0-logo/logo_start.ts
#EXT-X-DISCONTINUITY
#EXTINF:12
resources/videos/vp1-QR/QR05_1.ts
#EXT-X-DISCONTINUITY
#EXTINF:2
resources/videos/vp2-intro-fluide-glacial/EtPendantCeTempsLaEn1975_processed.ts
#EXT-X-DISCONTINUITY
#EXTINF:5
resources/videos/vp3-fluide-glacial/CarteAPuce_processed.ts
#EXT-X-DISCONTINUITY
#EXTINF:278
resources/videos/vp4-interviews/Anecdote03-Jean-PierreBantre_cut.ts
#EXT-X-DISCONTINUITY
#EXTINF:2
resources/videos/vp5-intro-affiches/EtPendantCeTempsLaDansLesCouloirsIrisa_processed.ts
#EXT-X-DISCONTINUITY
#EXTINF:10
resources/videos/vp6-affiches/mallet.ts
#EXT-X-DISCONTINUITY
#EXTINF:5
resources/videos/vp7-remerciements/generique_processed.ts
#EXT-X-DISCONTINUITY
#EXTINF:2
```

[3] Plusieurs méthodes sont en fait possibles pour lire des vidéos avec le lecteur Flowplayer (https://flowplayer.org/docs/playlist.html) mais on ne veut pas de coupure entre les vidéos, ce qui amène quelques spécificités techniques (e.g., mode discontinuité) et contraintes.

[4] Dans la suite, nous allons re-considérer le problème de la génération de la playlist dans le format “M3U étendu” et compréhensible par le lecteur Flowplayer. N’essayez donc pas d’implémenter une solution à ce moment là du TP

```
resources/videos/vp8-logo/logo-end.ts
#EXT-X-ENDLIST
```

Exemple de playlist M3U étendue, compréhensible par Flowplayer. #EXTINF:3 spécifie la durée de la vidéo (3 secondes); #EXT-X-DISCONTINUITY spécifie que les séquences doivent être jouées sans discontinuité; le format .ts est le format d'entrée exigé par le lecteur

Q6 : Quels étaient les bénéfices attendus lors du développement du métamodèle intermédiaire de playlist ? Quelles ont été les difficultés rencontrées ?
Discuter de votre expérience.
Proposer des recommandations sur l'utilisation d'un métamodèle intermédiaire, de transformations model-to-model, et de transformations model-to-text en vous basant sur votre expérience.

D'autres formats peuvent être considérées (pour d'autres lecteurs, ou simplement pour le lecteur Flowplayer lui-même, cf https://flowplayer.org/docs/playlist.html).
Dans la suite vous pourrez utiliser vos transformations model-to-model ou bien model-to-text -- libre à vous.

3. Augmenter une spécification VideoGen pour le lecteur Web

Objectifs: maîtrise de la transformation de modèles (model-to-model); refactoring; TP4

Une spécification VideoGen est parfois "incomplète", car fastidieuse à écrire. Une piste d'amélioration est alors d'automatiser le remplissage de ces informations.
Par exemple, la "durée" d'une séquence video peut être renseignée manuellement, mais un outil automatique pourrait le faire pour nous. Cette durée est notamment nécessaire pour générer certaines playlists (e.g., M3U étendue).

Q7 : Ecrire une transformation qui prend en entrée une spécification VideoGen et qui assigne une valeur « durée » pour chaque séquence vidéo. On utilisera ffmpeg pour retourner la durée d'une vidéo (lire par exemple: http://superuser.com/questions/650291/how-to-get-video-duration-in-seconds).

Q8 : Ecrire une transformation qui prend en entrée une spécification VideoGen "augmentée" et qui génère une playlist dans le format "M3U étendu", compréhensible par le lecteur web Flowplayer. Nous utiliserons une transformation model-to-text ici.
Générer également la page Web HTML pour effectivement jouer une variante de vidéo.

Le lecteur Flowplayer, en mode discontinuité, attend des vidéos au format ".ts". Il faut donc convertir chaque séquence vidéo en ".ts". On utilisera ffmpeg pour convertir une vidéo au format ts:

```
ffmpeg -i INPUT.mp4 -strict -2 -vcodec h264 -acodec aac -f mpegts OUTPUT.ts
```

Cependant il faut exécuter ce traitement sur toutes les séquences vidéos de la spécification VideoGen.

Q9 : Ecrire une transformation qui prend en entrée une spécification VideoGen et qui génère en sortie un ensemble de vidéos ".ts"

4. Vers un configurateur et des vignettes

Objectifs: maîtrise de la transformation de modèles (model-to-*); TP4

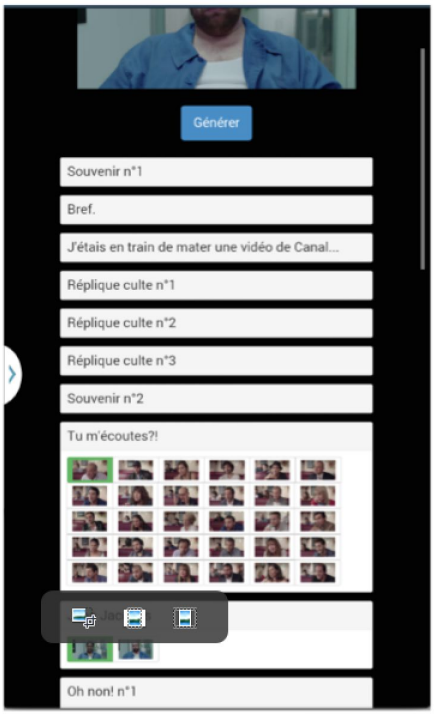
Plutôt que de générer aléatoirement des variantes de vidéos, on aimerait qu'un utilisateur ait le contrôle sur la génération et puisse sélectionner les séquences vidéo qu'il souhaite inclure.
Concrètement un utilisateur manipulerait alors une interface (un configurateur) où :

- Il dit « oui » ou « non » quand une séquence vidéo est optionnelle
- Il sélectionne une séquence vidéo dans le cas d'une alternative
- Pour les séquences obligatoires il ne peut rien faire

On considère qu'il est plus agréable pour l'utilisateur de visualiser une image (vignette) de la séquence vidéo lors des différents choix. Intuitivement cela permet de « résumer » une séquence vidéo avec une image.

Q10 : Ecrire une transformation qui prend en entrée une spécification VideoGen et qui génère en sortie un ensemble de vignettes pour toutes les séquences vidéos. On utilisera ffmpeg pour générer les vignettes. La commande est la suivante:
ffmpeg -y -i $video -r 1 -t 00:00:01 -ss 00:00:$2 -f image2 $dir/$name.png

Q11 : Ecrire une transformation qui prend en entrée une spécification VideoGen et qui génère en sortie une page Web affichant les vignettes. L'ordre d'affichage des vignettes sera déterminé par l'ordre de la spécification VideoGen. Pour les « alternatifs » les vignettes seront affichées dans un même bloc (cf exemple ci-dessous).



5. Vérification des spécifications VideoGen

Objectifs: maîtrise de la transformation de modèles (model-to-*); vérification; TP4/TP5

Une specification VideoGen peut contenir des erreurs – non pas syntaxiques car celles-ci sont par construction non autorisées par la grammaire mais plutôt liées à l'interprétation de certaines données.
Un exemple très concret est le suivant: si deux séquences video ont la même location, c'est certainement une erreur et on peut émettre un "Warning".
D'autres exemples:

- Les identifieurs doivent être uniques
- Les probabilités doivent être cohérentes (une probabilité d'une sequence vidéo "optionnelle" ne peut dépasser 1 (100%); la somme des probabilités ne doit pas dépasser 100% pour les sequences video "alternatif")

Certaines erreurs peuvent être gérées simplement en modifiant la grammaire Xtext. D'autres erreurs requièrent une visite globale du modèle VideoGen pour verifier la coherence des informations.

Q12 : Ecrire une transformation qui prend en entrée une spécification VideoGen et qui vérifie un certain nombre de propriétés ci-dessus

Cette étape de vérification peut s'intégrer de différentes manières, par exemple, avant l'appel concernant les transformations de modèle précédentes – on peut faire le parallèle avec un compilateur qui émet des messages d'avertissements (Warning) ou qui stoppe le processus si l'erreur est considérée comme sévère.

Q13 (optionnel) : Etudier Xtext et intégrer cette étape de vérification pour que des messages d'erreurs ou des avertissements apparaissent dans l'éditeur lorsque l'utilisateur écrit une spécification VideoGen ne respectant pas les propriétés mentionnées ci-dessus.

6. Un site Web complet (Putting all together)

Objectifs: maîtrise de la transformation de modèles; intégration; Web; TP5/TP6

Grâce aux transformations précédentes on est presque en mesure de déployer VideoGen.
Il manque quelques facilités au niveau de l'interface Web.
On utilisera Jhipster (http://jhipster.github.io) pour développer le site Web.

Lors de la generation aléatoire de variantes, on aimerait plus de dynamisme et une solution Web dans laquelle un utilisateur clique sur un bouton "Generate" pour re-lancer la génération d'une nouvelle variante de video.

Q14 : Implémenter une telle solution en effectuant une requête à un serveur Web (via JavaScript) lors du clique du bouton « Generate » ; le serveur génèrerait alors une nouvelle playlist aléatoirement.

On s'intéresse maintenant au mode « configurateur ». A partir des choix de l'utilisateur, on souhaite effectivement jouer la variante de vidéo correspondante.

Q15 : Implémenter une solution en effectuant une requête à un serveur Web (via JavaScript) qui génèrerait alors la playlist correspondante aux choix de l'utilisateur.

7. Paramétriser VideoGen

Objectifs: maîtrise des techniques de variabilité (modélisation, implémentation); TP6

On approche d'une solution complete, où, à partir d'une specification VideoGen, on est en mesure de déployer un site Web capable de générer des variantes de videos.

On constate cependant que deux modes ont été considérées jusque-là: le mode “aléatoire” et le mode “configurateur” – c’est soit l’un, soit l’autre. Aussi notre générateur VideoGen est potentiellement paramétrisable. En fait il y a plus de paramètres que ces deux modes là. Ci-dessous un texte (en français, brut de décoffrage) décrit informellement les différents points de variation de VideoGen.

“Il y a cinq modes: le mode aléatoire, le mode avec le support des probabilités, le mode configurateur, le mode configurateur hybride, et le mode crazy,.
Le mode aléatoire est bien connu et déjà implémenté: c’est un tirage aléatoire sur les optionelles et alternatifs. Le mode avec les probabilités tient compte des fréquences d’apparition des sequences. Le mode configurateur est bien connu et déjà implémenté mais c’est un peu fastidieux de sélectionner chaque sequence video: on souhaite ne faire configurer que certaines sequences videos, et pour le reste: vive l’aléatoire ou les probabilités!
Le mode crazy assemble les sequences videos dans un ordre aléatoire (avec ou sans configuration, avec ou sans probabilités).

La possibilité de sauvegarder les videos? Hum cela depend des cas – ce peut être ineffectif pour des raisons de copyright sur les données; ça coûte aussi en bande passante et temps de calcul.
Une autre fonctionnalité intéressante est de pouvoir afficher une vignette de la sequence video en cours de lecture. Mais c’est clairement optionnel et uniquement quand on n’est pas en mode configurateur”

Q16: Formaliser les exigences sous forme d’un feature model. L’idée est qu’à partir d’une configuration du feature model (eg mode crazy sans sauvegarde mais avec visualisation des vignettes) on déploie un nouveau site Web avec Jhipster.
Vous utiliserez FAMILIAR et son DSL interne ou externe (cf Annexe)

Q17: Démontrer que vous êtes dores et déjà capable de déployer un Jhipster / VideoGen en fonction de certaines configurations valides de votre feature model.

Q18: Implémenter une « feature » supplémentaire parmis celles non implémentées (eg possibilité de sauvegarder une vidéo dans le site Web, à la fin de la lecture d’une variante de vidéo). Décrire les configurations supportées et celles qui ne le sont pas.

8. Evaluation

Dans le cadre du contrôle continu, vous avez à rendre un projet mettant en œuvre tous les concepts et toutes les technologies abordés en cours, à savoir :

- Métamodélisation avec Ecore/EMF
- Spécification de DSL externe avec Xtext
- Transformation de modèles (model-to-model, model-to-text) avec Xtend
- Gestion de la variabilité
- Diverses piles technologiques Web

Le projet consiste à adresser toutes les questions. Il n'est pas attendu des réponses à ces questions (donc pas de rapport!), mais une implémentation montrant que vous êtes en mesure d'y répondre.
Pour réaliser et rendre le travail, vous utiliserez un repository Git. Ce repository contiendra nécessairement:

- Une grammaire/un projet Xtext pour le DSL de générateur de vidéos
- Les différentes transformations écrites en Xtend
- Des instructions en anglais dans un README.md sur le projet (motivation, technologies, architecture, déploiement, etc.)
- Une réponse à la Q16 pour documenter l'état du projet
- Un screencast de démonstration

Le travail est à rendre pour le 19 décembre 2015. Un email mentionant l'adresse du repository est à envoyer à mathieu.acher@irisa.fr

Vous pouvez vous partager le travail: les groupes de 2 sont autorisés.

Annexe

Flowplayer et M3U étendu

https://flowplayer.org/

```
<a id="PLAYERID"></a>
```

```
flowplayer("PLAYERID", "PATH/TO/flowplayer.swf", {
  wmode: 'direct',
  plugins: {
    httpstreaming: {
      url: 'PATH/TO/flashlsF
      hls_live_flushurlcache : false,
      hls_seekmode : "ACCURATE",
      hls_fragmentloadmaxretry : -1,
      hls_manifestloadmaxretry : -1,
      hls_capleveltostage : false,
```

```
        hls_maxlevelcappingmode : "downscale"
      }
    },
    clip: {
      accelerated: true,
      url: "PATH/TO/PLAYLIST.m3u",
      urlResolvers: "httpstreaming",
      lang: "fr",
      provider: "httpstreaming",
      autoPlay: false,
      autoBuffering: true
    },
    log: {
      level: 'none',
      filter: 'org.flowplayer.controller.*'
    }
});
lowPlayer.swf',
        hls_debug : false,
        hls_debug2 : false,
        hls_lowbufferlength : 3,
        hls_minbufferlength : 5,
        hls_maxbufferlength : 0,
        hls_startfromlevel : -1,
        hls_seekfromlevel : -1,
```

Attention: il faut installer les bons codecs, et toutes les vidéos doivent être préalablement au format “.ts”
(
vous pourrez utiliser ffmpeg en amont pour la conversion en “ts” si nécessaire:
ffmpeg -i INPUT.mp4 -strict -2 -vcodec h264 -acodec aac -f mpegts OUTPUT.ts
)

FAMILIAR

http://familiar-project.github.io/
http://mathieuacher.com/teaching/MDI/fmlapp-1.0-SNAPSHOT.tgz

cf documentation sur DSL externe

(DSL interne)
FeatureModelVariable fm1 = FM (“A : B [C] [D]; C -> D; ”);
ConfigurationVariable cf1 = …
...
cf1.getSelectedFeatures();

Exercices supplémentaires / boîte à idée (hors évaluation)

Il est parfois utile de pouvoir manipuler et référencer un “identifieur” pour une sequence video donnée. Certaines sequences n’ont pas d’identifieur, car c’est fastidieux à écrire.

Q: Ecrire une transformation qui prend en entrée une spécification VideoGen et qui assigne des identifieurs uniques pour chaque séquence vidéo qui n’en ont pas.

====

L’écriture d’une spécification VideoGen peut prendre énormément de temps et est sujet à erreur. Aussi on propose d’écrire automatiquement une spécification VideoGen à partir d’un dossier de vidéos qui suit une organisation/structure spécifique. On suit en quelque sorte le principe de “convention over configuration”
https://en.wikipedia.org/wiki/Convention_over_configuration
Cette écriture automatique est notamment intéressante pour “commencer” un projet VideoGen puisque les utilisateurs collectent et organisent les vidéos.
Aussi, en suivant une certaine structure pour organiser les vidéos dans des dossiers, il pourrait obtenir une spécification VideoGen qu’il pourrait éditer par la suite pour jouer sur la variabilité, les fréquences d’apparition, etc.
Le dossier aurait la structure suivante:

```
/
  v1/
  v2/
  v3/
  ..
  vN
```

chaque dossier (v1, v2, …, vN) contient des séquences vidéos… S’il y a une seule séquence dans le dossier, alors c’est une vidéo “obligatoire”. Sinon c’est une séquence “alternative”. L’ordre est caractérisée par le numéro de dossier.

Dans le sens inverse, on peut très certainement envisager la re-structuration (en “export”) des vidéos dans un dossier cible (en suivant la structure décrite) à partir d’une spécification VideoGen.

====

Un langage plus riche avec répétition, variable, “composite”, mécanismes pour modulariser.

====

D’autres players, d’autres artefacts (mp3)

====

Social video: possibilité de “liker” des variantes de vidéo, de commenter, etc.