OSSIA : OPEN SCENARIO SYSTEM FOR INTERACTIVE APPLICATIONS

Théo De La Hogue — GMEA — theod@gmea.net

Pascal Baltazar — L'Arboretum — pascal@baltazars.org

Myriam Desainte Catherine — myriam.desainte-catherine@labri.fr

Jaime Chao — LaBRI — jaimitochao@gmail.com

Clément Bossut — bossut.clement@gmail.com

RÉSUMÉ

Cet article fait état des travaux réalisés dans le cadre du projet Open Scenario System for Interactive Application (OSSIA) dont l'objectif est d'offrir des outils génériques pour le développement de logiciels d'écriture de scénario interactif. Après un rappel des acquis sur lesquels le projet s'appuie, nous présenterons les méthodes adoptées pour adapter le formalisme du moteur d'écriture logico-temporel à des situations réelles de production et décrirons un exemple d'implémentation à travers une nouvelle version du logiciel *i-score*.

DES ENJEUX TRANSDISCIPLINAIRES

Qu'il s'agisse de dispositifs interactifs, de jeux vidéo, du spectacle vivant, de l'industrie culturelle ou de la muséographie, la mise en jeu de différents médias requiert l'écriture d'un scénario régissant l'ensemble des interactions. Cette écriture spécifique définissant les comportements de contenus numériques au sein d'un dispositif informatique, en fonction d'un contexte matériel ou numérique, reste cependant une opération complexe et réservée aux experts : la mise en œuvre au sein d'un environnement de programmation intermédia d'une intention scénaristique, même simple, nécessite en effet de prendre en compte l'ensemble des possibles du dispositif et de son environnement. Cette scénarisation dont le déroulement est ouvert, relève de la programmation et est très peu assistée. De plus, la moindre modification peut être source d'aberration et induire des dysfonctionnements de l'ensemble du système.

Pourtant, la description de tels systèmes par leurs concepteurs ou leurs utilisateurs non programmeurs ne reflète pas cette complexité. Elle semble hiérarchisée intuitivement en unités d'actions parfois séquencées, parfois conditionnées au sein d'un schéma mêlant les entrées et sorties du système afin de tenir compte du contexte ou de le modifier. Le projet Open Scenario System for Interactive Application (OSSIA) vise ainsi à favoriser l'écriture de tels scénarios en proposant un ensemble d'outils logiciels autorisant l'édition et l'exécution d'une description schématisée d'un système automatisé.

Reçu dans le cadre de l'appel à projet ANR 2012 Contenu et Interaction, le projet OSSIA[1] réuni un consortium d'acteurs issus de la recherche institutionnelle ou privée, de l'industrie ou œuvrant dans le domaine de la création : le Laboratoire Bordelais de Recherche en Informatique (LaBRI) formalise et développe le moteur logico-temporel, le GMEA – Centre National de Création Musicale d'Albi-Tarn coordonne le projet et structure le partage des outils, le laboratoire Cédric et l'École Nationale du Jeu et des Médias Interactifs (ENJMIN) du CNAM et les sociétés Blue Yeti et RSF intègrent respectivement les outils d'écriture pour la réalisation de scénario interactifs dans les jeux vidéo, dans des dispositifs muséographiques et dans du matériel de pilotage embarqué. OSSIA s'inscrit par ailleurs dans un réseau de nombreux collaborateurs liés à la création ou encore à la formation tels que l'Institut Supérieur des Techniques du Spectacle (ISTS) et l'Ecole Nationale Supérieure d'Arts et Techniques du Théâtre (ENSATT).

CONTEXTE DE RECHERCHE

Cette dynamique de recherche s'inscrit dans des réflexions sur les usages dont certaines conclusions, qu'il nous semble important de rappeler dans cet article, ont esquissé des spécifications pour le projet OSSIA.

2.1. Historique

C'est à l'initiative de la Compagnie Incidents Mémorable/didascalie.net que l'Association Française d'Informatique Musicale (AFIM) a constitué en 2006 un groupe de travail pour mener une étude [1], visant à mettre en œuvre un questionnement des pratiques, des outils et des métiers du sonore dans le contexte du spectacle vivant (théâtre, danse, concert étendu ou multimédia) pour en dégager les particularités. L'objectif était d'établir un état de l'art des environnements logiciels existants pour la composition et l'interprétation du sonore dans le cadre du spectacle vivant et de dégager les prospectives de développement futur de tels environnements. La conclusion qu'une mutualisation de la recherche vers le développement d'interfaces numériques de création multimédia était

[1] Référence ANR-12-CORD-0024

nécessaire a impulsé la création de la plateforme Virage en 2008.

Missionné par l'Agence Nationale de la Recherche, la Plateforme Virage[2] avait pour objectifs de définir un cahier des charges et des spécifications générales pour des développements futurs de nouvelles interfaces de contrôle et d'écriture de contenus multimédia temps-réel (son, image, lumière, machineries, ...) pour la création artistique et les industries culturelles. Suite à une étude approfondie des usages, pratiques et besoins des professionnels de ces domaines [2], le projet a débouché sur de nombreux prototypes (dont le séquenceur Virage) et sur le constat qu'une structure pour partager les développements informatiques permettrait une dynamique de travail en réseau :

> *« La conclusion principale de cette étude des usages a confirmé la nécessité de travailler autour des notions d'interopérabilité et de modularité, en visant la combinaison de nos développements avec les environnements logiciels et matériels existants et correspondant aux habitudes des praticiens, plutôt que l'intégration de cette réalité complexe et multiple dans un illusoire logiciel unique et certainement trop polyvalent pour être viable. »* [3]

2.2. La réalisation de dispositif intermédia

Afin de bien cerner les enjeux d'une recherche au sujet des outils d'écriture, voici un état des lieux des outils actuels propres à la réalisation de dispositifs constitués de plusieurs médias interconnectés dont il faut assurer le contrôle et/ou en écrire le comportement.

Les différents éléments à notre disposition sont :

- des *environnements de création* : Live, VDMX, DLigth, Reaktor, Modul8, MadMapper, adobe Flash, Unity3D, *etc.*,
- des *environnements de programmation* : Max, Pure Data, Isadora, Arduino, Processing, *etc.*,
- des *interfaces de contrôle et de captation* : surface tactile, Kinect, capteur posturologique/physiologique, caméra, micro, BCF2000, MPD24, Monome, *etc.*,
- et du *matériel de diffusion contrôlable ou non* : projecteur de lumière, haut-parleur, moteur, vidéo-projeteur, *etc*.

[2] www.virage-platform.org

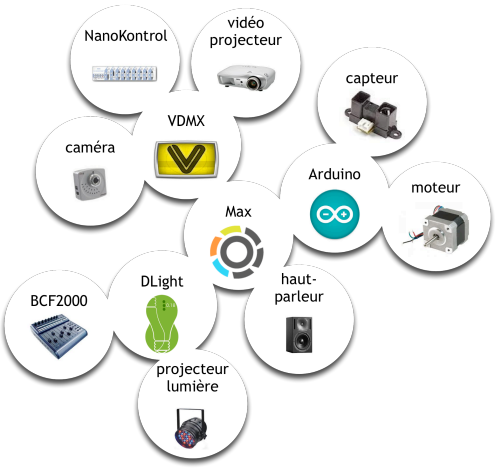

Figure 1. Exemple de dispositif intermédia

2.3. Les moteurs d'écritures

Si les environnements informatiques offrent de réel potentiels d'interconnections de médias, l'écriture du comportement dans le temps de ces éléments interconnectés ramène souvent le réalisateur intermédia à un travail de programmation préalable ou conjoint au travail de création.

Parfois les moteurs d'écriture sont propre à l'environnement de création (Live, VDMX, DLight) tandis que d'autres solutions assurent un contrôle externe (Qlab, Duration, Vezér, i-score, Yannix). Mais certains projets nécessitent le développement d'un moteur d'écriture spécifique, adapté à des exigences esthétiques, à des contraintes techniques ou à des utilisateurs non-experts. Dès lors, les compétences requises pour architecturer correctement un moteur d'écriture se rapprochent de l'ingénierie logicielle (gestion de base de données pour adresser les ressources, mémoriser et rappeler leurs états, programmation d'automate, programmation concurrente, *etc.*).

Ainsi ces constats ont motivé la conception d'une librairie logicielle au service de développements spécifiques de moteurs d'écritures de scénarios interactifs.

METHODOLOGIE

La conception d'éléments logiciels génériques et réutilisables de manière spécifique implique des architectures modulaires et interopérables. Dans cet objectif, le projet OSSIA adopte une démarche de partage des outils et les expérimente en situation de production.

3.1. La structuration du partage

OSSIA s'appuie sur une organisation de développement informatique collaboratif international et open

source (Jamoma[3]) pour constituer un base commune d'éléments logiciels[4]. Les produits de la recherche sont hétéroclites et regroupent une API[5] en cours d'écriture, la librairie C++ Score, dont le formalisme est décrit plus bas, et divers *externals* ou *patchs* Max issus de préoccupations techniques diverses mais intégrables dans différents domaines d'application spécifiques. À noter que la problématique de l'interopérabilité, dont il n'est pas question dans cet article, représente une part non-négligeable des éléments partagés au sein du projet. Pour chaque chantier, nous nous assurons en premier lieu qu'il est possible aux moteurs d'écritures de dialoguer avec les logiciels ou environnements de travail propres à chaque domaine d'application.[6]

3.2. Les chantiers d'expérimentation

Lors des chantiers d'expérimentations en situation réelle de production, des retours concrets sur l'utilisation des notions et outils issus de la recherche viennent alimenter les formalismes sous-jacents.

Durant ces échanges autour de la question des scénarios interactifs la communication entre experts est essentielle mais les discussions peuvent se heurter au fait qu'ils ne disposent pas d'un langage unique comme moyen d'échange [4]. Ainsi une médiation est nécessaire entre les différents acteurs afin de vérifier la robustesse des modèles établis par la recherche face à des cas réels d'utilisation et c'est pourquoi le projet OSSIA s'attache à proposer un formalisme d'écriture de scénarios interactifs.

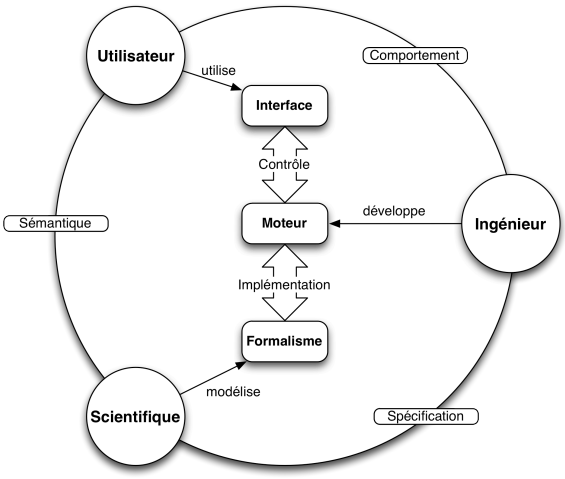

Figure 2. Relations entre les différents acteurs à travers les chantiers d'expérimentations

[3] www.jamoma.org propose des solutions fondées sur les besoins issus des pratiques de la création artistique.

[4] github.com/OSSIA

[5] github.com/OSSIA/API

[6] À titre d'exemple les environnements Live, Modul8 et Unity3D font l'objet d'un effort particulier pour être rendu interopérable en vue d'une utilisation lors des chantiers d'expérimentation.

VERS UN FORMALISME D'ECRITURE

Les travaux menés dans le cadre du projet OSSIA définissent par « scénario interactif » une structure contenant potentiellement du temps souple, conditionnel, non-linéaire et hiérarchique. Ces notions renvoient aux possibilités de contrôler la vitesse d'exécution, d'attendre ou d'ignorer le déclenchement de certaines parties, d'en répéter l'exécution ou d'autoriser des choix entre plusieurs d'entre elles. La hiérarchie renvoie à des besoins d'organisation en sous-partie d'une structure temporelle.

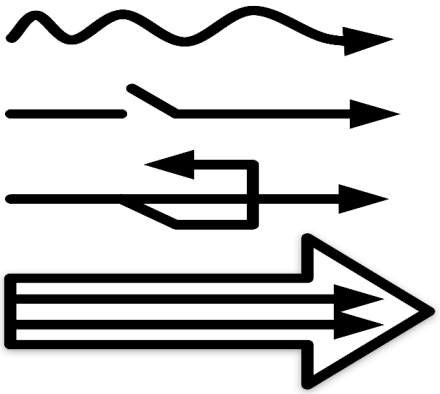

Figure 3. Temps Souple, Conditionnel, Non-Linéaire, Hiérarchique

4.1. Modélisation du temps

La formalisation informatique de ces propriétés temporelles s'appuie sur le modèle existant de partitions interactives [5] et consiste à l'étendre aux conditions, à permettre les répétitions [6] et à autoriser une structuration hiérarchique (propriété modélisée récemment au moyen du langage RML [7]).

Parallèlement, la conception de la librairie *Score* a pris le parti de revisiter les accès au formalisme proposé dans *libIscore* [8][9] à la lumière des retours de l'expérimentation du séquenceur Virage [10]. Le nouveau formalisme s'appose en quelque sorte au-dessus de l'ancien formalisme pour le réorganiser à travers des éléments modulaires plus intuitifs à manipuler ; *Score* considérant par exemple les processus temporels comme encadrés par des évènements eux-mêmes partageables entre processus (l'évènement de fin d'un processus pouvant être l'évènement de début d'un autre processus par exemple) là où *libIscore* ne considérait que des durées possédant chacune leur début et leur fin.

4.2. Lexique

En premier lieu il est nécessaire d'établir un lexique pour décrire les éléments constitutifs d'un scénario interactif :

- *Service* : ce qu'offre une ressource pour son utilisation. Nous distinguons :
  - le *paramètre* dont l'état est susceptible d'être mémorisé (ex : le volume d'une piste audio),
  - le *message* dont l'état ne peut être mémorisé (ex : le déclenchement de la lecture),
  - le *retour* dont l'état ne peut pas être contrôlé (ex : l'enveloppe de l'amplitude ou la fin de lecture).
- *État* : la valeur d'un ou plusieurs services.
- *Namespace* : l'ensemble des noms des services organisés sous la forme d'une arborescence. Par exemple :
  /player
  /play
  /volume
  /end
- *Actions* : une opération faites sur l'état
  - *set* : mise à jour de l'état.
  - *get* : demande de l'état.
  - *listen* : écoute de l'état.
- *Condition* : vérification effectuée sur l'état d'un service. Par exemple on peut vérifier l'expression « /player/volume > 12 ».
- *Événement* : un point dans le temps déclenché à une date précise ou à la validation d'une condition. Des actions peuvent lui être associées.
- *Processus* : une opération effectuée pendant un temps qui peut être fixe, borné ou libre selon la nature de l'évènement qui marque sa fin :
  - *Intervalle* : une quantité de temps qui peut être fixe, bornée ou libre.
  - *Automation* : met à jour l'état d'un service dans le temps selon une courbe.
  - *Mapping* : écoute de l'état d'un service pour mettre à jour l'état d'un autre service.
  - *Scénario* : contient et supervise des évènements, des conditions et des processus.
  - *autres* : d'autres types de processus sont envisageables (ex : générateur).

4.3. Application à un cas réel d'installation

Nous collectons un maximum de cas réels d'utilisation afin de vérifier la pertinence du formalisme. À titre d'exemple le formalisme de *Score* est ici utilisé pour décrire le comportement d'une installation réalisée par Abtin Sarabi, étudiant à l'Institut Supérieur Des Arts de Toulouse (ISDAT).

En premier lieu voici une description textuelle du comportement de son installation : *le visage d'un prêtre est affiché sur une télévision. Tant que personne ne marche devant l'écran, plusieurs films montrent combien le prêtre s'ennuie (il regarde le ciel, sa montre, etc.). Mais si une personne marche devant l'écran (et que le film le montrant en train de s'ennuyer se termine), le prêtre commence à regarder la personne fixement. Ses yeux suivent la personne pour attirer son attention. Puis, lorsque la personne s'en va, le prêtre s'ennuie à nouveau.*

Afin de bien cerner la mise en œuvre de cette installation et donc l'écriture de son comportement, en voici la description des ressources : il y a un lecteur de film, une banque de films, un système de détection par caméra qui retourne s'il y a une présence et la position de cette présence. Nous considérons que l'état de ces ressources peut être adressé, mémorisé, rappelé ou écouté via des solutions protocoles.

Le *namespace* des ressources se résume ainsi :
/bank
/select : message pour sélectionner un film.

/movie
/play : message pour lancer le film.
/position : paramètre pour positionner la tête de lecture proportionnellement à la taille du film.
/end : retour avertissant que le film est terminé.

/detection
/presence : retour de détection d'une présence.
/position : retour de la position de la présence.

Dès lors nous avons tout le nécessaire pour formaliser avec *Score* la description textuelle du comportement de l'installation[7] :

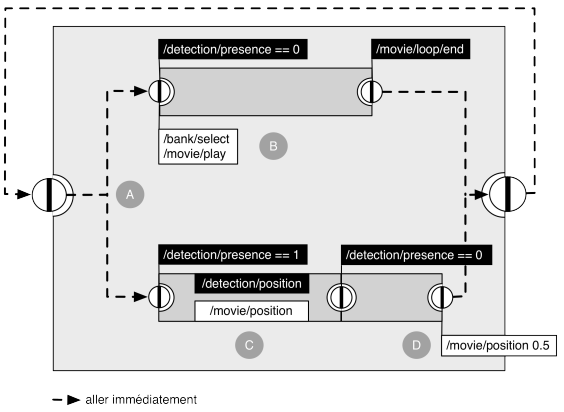

Figure 4. Scénario du cas réel d'installation formalisé

[7] Cette représentation graphique est amenée à évoluer à mesure que sa manipulation, pour modéliser des cas réels de scénario interactifs dans différents domaines, en montre les limites.

Le scénario de la figure 4 peut se traduire comme suit :

- A : Deux évènements interactifs sont conditionnés par l'état de */detection/presence* (*get*) pour sélectionner soit B (si égal à 0) soit C (si égal à 1).
- B : Sélectionne un autre film avec le message */bank/select* (*set*) et le lance en utilisant le message */movie/*play (*set*). Puis le retour */movie/end* est attendu (*listen*) avant de reboucler sur A.
- C : Mets en lien le retour */detection/position* (*listen*) sur le paramètre */movie/position* (*set*). Le mapping est actif jusqu'à ce que */detection/presence* devienne 0 (*listen*). Puis on passe à D.
- D : Ramène le paramètre */movie/position* depuis sa position courante (*get*) à 0.5 (*set*) en un temps précis avant de reboucler sur A.

Cette formalisation du comportement de l'installation prend tout son sens comparée à son implémentation dans l'environnement de programmation Max :

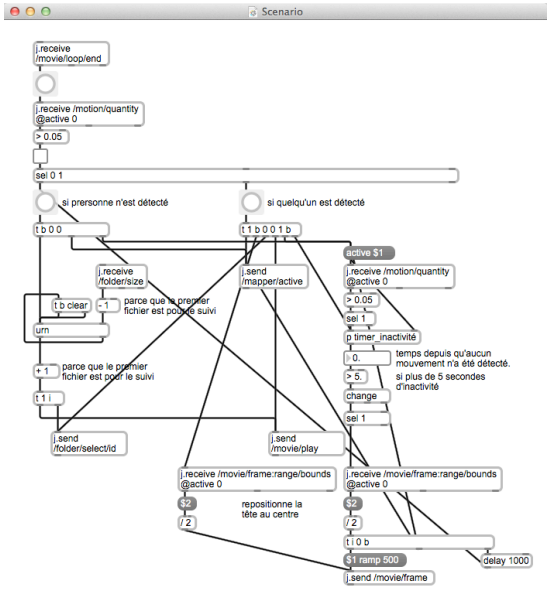

Figure 2. Scénario du cas réel d'installation dans Max

D'un point de vue syntaxique, *Score* offre donc la possibilité d'exprimer un même comportement logico-temporel en convoquant beaucoup moins d'éléments que d'autres langages de programmation. Néanmoins cette représentation graphique du formalisme n'est pas satisfaisante d'un point de vue ergonomique. La présentation ci-après du logiciel *i-score* montre un exemple d'application du formalisme de *Score* dans une manipulation de type séquenceur.

ARCHITECTURE DE LA LIBRAIRIE SCORE

La libraire *Score*[8] est organisée en quatre classes principales : *TimeEvent*, *TimeCondition*, *TimeProcess* et *TimeContainer*.

5.1 La classe *TimeEvent*

La classe *TimeEvent* possède une date indicative, un statut (*waiting*, *pending*, *happened*, *disposed*) et peut référencer des mémorisations d'états à rappeler lors de son activation. Un *TimeEvent* connait la *TimeCondition* à laquelle il est soumis ainsi que le *TimeContainer* auquel il appartient.

5.2 La classe *TimeCondition*

La classe *TimeCondition* possède une table d'évènements classé par *Expression* à évaluer pour valider l'activation ou non de chaque l'évènement. Une *TimeCondition* peut être prête ou non à s'évaluer, selon que tous les évènements qu'elle supervise sont *pending* ou non. La classe connait le *TimeContainer* auquel elle appartient.

5.3 La classe *TimeProcess*

La classe *TimeProcess* définit une interface pour la compilation, le démarrage, l'exécution et la terminaison d'un contenu temporel quelconque. Un *TimeProcess* se lie à deux *TimeEvent* (associés à son début et à sa fin) dont il observe le statut *happened* ou *disposed*. Lorsque le statut de son événement de début devient *happened*, un *TimeProcess* active une unité d'horloge et appelle une méthode virtuelle *ProcessStart* implémentée dans chacun des *plugins* de *TimeProcess*. À chaque pas de progression de l'unité d'horloge un *TimeProcess* appelle une méthode virtuelle *Process* implémentée dans chacun des *plugins* de *TimeProcess* et où la valeur de la progression est utilisée de manière spécifique. Lorsque le statut de son événement de fin devient *happened*, un *TimeProcess* désactive son unité de d'horloge et appelle une méthode virtuelle *ProcessEnd* implémentée dans chacun des *plugins* de *TimeProcess*.

À noter que le type d'horloge est sélectionnable selon que l'on souhaite se référer à l'horloge du système ou à une horloge externe par un système de *plugin* permettant de créer tout type d'horloge. Par exemple nous prévoyons la possibilité de synchroniser l'exécution via le réseau pour une exécution distribuée sur plusieurs ordinateurs.

La classe *TimeProcess* peut aussi définir une durée minimum et maximum au cas où la fin de son exécution soit déterminée par un *TimeEvent* soumis à une *TimeCondition* (bien que d'autres cas puissent aussi conduire à définir ces bornes).

Il existe pour l'instant deux *plugins* de *TimePro-*

[8] https://github.com/OSSIA/Score

cess : *Intervalle* et *Automation*. Nous prévoyons le développement d'un *plugin* de *mapping*, de générateurs et le système permet d'envisager le développement de *plugins* dédiés à la lecture de média.

5.4 La classe *TimeContainer*

La classe *TimeContainer* hérite de *TimeProcess*. La classe définie une interface pour l'ajout, la suppression et la supervision d'actions sur des listes de *TimeEvent*, *TimeCondition* et *TimeProcess*.

Il existe pour l'instant un seul *plugin* de *TimeContainer* : *Scenario* qui offre un gestionnaire de contrainte pour conserver les relations de précédence entre évènements lors de l'édition et compile un réseau de Petri pour en vérifier l'exécution.

IMPLEMENTATION DANS I-SCORE

Depuis octobre 2013 une équipe de développement du LaBRI entreprend une refonte complète du logiciel *i-score*[9] afin de mieux tirer partie des propriétés du formalisme proposé par Score tout en offrant plus de confort à l'utilisateur. Nous présentons ici le résultat de réflexions qui sont en cours d'implémentation dans une nouvelle version d'*i-score*.

6.1 Une représentation modulaire

Le logiciel *i-score* est une interface graphique permettant à l'utilisateur de manipuler simplement un scénario logico-temporel pendant ses phases d'édition ou d'exécution. L'utilisateur peut écrire un scénario en plaçant et en manipulant des objets graphiques sur une ligne de temps s'écoulant de la gauche vers la droite.

L'édition consiste en premier lieu à placer des évènements temporels dans le scénario (Figure 6.). Chaque évènement temporel (a) peut posséder un état (b) ou plusieurs (c), une interaction (d) ou plusieurs (e). Un état peut être lié à une interaction (f). A noter que l'ajout de plusieurs interactions autorise l'édition de choix ; lorsqu'une interaction est validée seul les éléments situés directement sous son influence sont exécutés.

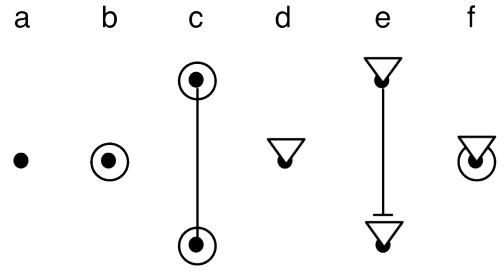

Figure 6. Évènement, état et interaction

[9] https://github.com/OSSIA/i-score

Par la suite un processus *intervalle*, symbolisé par une barre horizontale (Figure 7.), peut être inséré entre une paire d'évènements ; déclarant par là même une relation de précédence entre eux et ainsi une contrainte lors du déplacement de l'un des deux. A priori la durée est fixe (a) mais si une interaction est insérée sur l'événement de fin, la durée devient souple et l'interaction sera possible n'importe quand dès que l'événement de début sera passé (b). Afin de préciser une période d'interaction, il est possible de faire apparaitre une *fourchette* de réglage. La borne gauche de la *fourchette* permet de préciser à partir de quand l'interaction peut arriver (c, d). La borne droite de la *fourchette* permet de préciser quand l'interaction devra être déclenchée si elle n'arrivait pas (d) ; l'absence de borne droite signifiant que l'interaction peut être attendue indéfiniment (c).

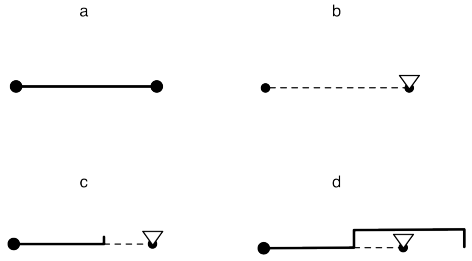

Figure 7. Processus, évènements et interaction

Un *intervalle* peut être transformé en un objet temporel plus complexe (Figure 8.). L'objet TimeBox peut contenir plusieurs processus partageant les mêmes évènements de début et de fin, et permettre de se replier sur son en-tête (b), ou de se dérouler sur plusieurs étages pour éditer le contenu de chacun de ses processus (c). Chaque *plugin* de processus possède une ou plusieurs représentations ; l'ensemble des représentations étant aussi architecturé en *plugin*. Dans le cas du processus *automation* plusieurs représentations sont envisagées selon le type du service à manipuler. Par exemple, une interface pour dessiner une trajectoire en deux dimensions sera préférable pour manipuler la conduite d'un paramètre de position, pour un paramètre de couleur une succession de gradient pourra être plus adapté, *etc*.

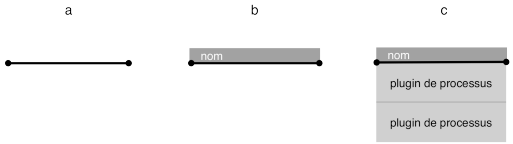

Figure 8. TimeBox

Pour une organisation hiérarchique du temps, un scénario est aussi un processus dans lequel tous les éléments graphiques présentés peuvent aussi être combinés au sein d'un sous-scénario (Figure 9.).

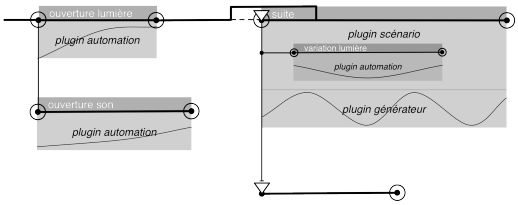

Figure 9. Exemple de scenario dans *i-score*

6.2. Prospections

Toutes les propriétés temporelles ne sont pas encore représentées dans *i-score* et font encore l'objet de discussions. Le cas des choix, lorsque que plusieurs interactions sont ajoutées à un événement, pose la question de la représentation du choix par défaut et d'une période de simultanéité durant laquelle plusieurs informations extérieures peuvent être attendues pour évaluer plusieurs interactions en même temps. La représentation des boucles reste aussi un problème. Par exemple, comment indiquer graphiquement l'équivalent d'un *do while* ou d'une boucle *for* ? Pour tous ces problèmes nous devons interroger les usages afin de déceler les démarches qui interviennent dans une écriture convoquant des choix ou des itérations.

CONCLUSION

Avec un moteur d'édition et d'exécution de scénario interactifs non dépendant des interfaces graphiques d'autres représentations que celle d'*i-score* sont imaginables selon les contextes d'utilisation et habitudes de travail : une représentation par *cuelist* peut être plus appropriée dans un contexte de spectacle vivant, un accès purement logique être souhaité dans un contexte muséographique tandis que certains outils de développement de jeux vidéos adopte une représentations spatiales pour des scénarisations non-linéaires. Cette indépendance permet aussi d'envisager qu'une fois écrit, un scénario interactif puisse être exécuté sur une plateforme plus légère sans soucier de sa représentation.

C'est pour toutes ces raisons que le projet OSSIA s'attache à rendre le formalisme de *Score* ouvert d'un point de vue logiciel en proposant une architecture de *plugin* pour les processus et les contenant temporel et qu'une API est en cours de rédaction. Grâce à un formalisme partagé et expérimenté dans des contextes de développements variés[10] tels que les jeux-vidéos, le spectacle vivant ou la muséographie, nous espérons proposer une librairie unifiée, générique et suffisamment souple pour répondre aux besoins spécifiques des écritures intermédias.

[10] Une implémentation de *Score* dans Max est prévue. Nous espérons ainsi permettre de créer rapidement des moteurs d'écritures spécifiques.

REFERENCES

[1] Baltazar, P., Gagneré, G. « Outils et pratiques du sonore dans le spectacle vivant », Actes des 12e Journées d'Informatique Musicale, Lyon, avril 2007. http://www.afim-asso.org/IMG/pdf/GT_son_spect_viv.pdf

[2] Santini, A., Sèdes, A., Simon, B. « Virage : Analyse des usages/État de l'art », rapport interne, version n° 3, Paris, 2009.

[3] Baltazar, P., Allombert, A., Marczak, R., Couturier, J-M., Roy, M., Sèdes, A., Desainte-Catherine, M. « Virage : Une réflexion pluridisciplinaire autour du temps dans la création numérique », Actes des 14e Journées d'Informatique Musicale, Grenoble, 2009.

[4] Meyssonier T. « Analyse des relations entre création artistique, modélisation mathématique et implémentation logicielle dans la problématique de l'écriture de structures temporelles », Rapport de stage effectué au LaBRI sous la direction de Desainte-Catherine M., Bordeaux, 2013.

[5] Allombert, A., Desainte-Catherine, M., Laralde J., Assayag, G. « A System of Interactive Scores Based on Qualitative and Quantitative Temporal Constraints », p1-8, Proceedings of ARTECH the Fourth International Conference on Digital Arts, Porto, Portugal, 2008.

[6] Toro, M., Desainte-Catherine, M., « Concurrent Constraint Conditional Branching Interactive Scores », p270-273, Proceedings of Sound and Music Computing, Barcelone, Espagne, Juillet 2010.

[7] Arias, J., Desainte-Catherine, M., Salvati, S., Rueda, C., « Executing Hierarchical Interactive Scores in Reactive ML », Journées d'Informatique Musicale, Bourges, Mai 2014, en soumission.

[8] Desainte-Catherine, M., Allombert, A., Assayag, G. « Towards a Hybrid Temporal Paradigm for Musical Composition and Performance : The Case of Musical Interpretation », p61-72, Vol. 37, No. 2, Computer Music Journal Summer, 2013.

[9] Marczak, R., Desainte-Catherine, M., Allombert, A. « Real-Time Temporal Control of Musical Processes », in Proceedings of the Third International Conferences on Advances in Multimedia, Budapest, Hongrie, Avril 2011.

[10] Allombert, A., Marczak, R., Desainte-Catherine, M., Baltazar, P., Garnier, L. « Virage : Designing an interactive intermedia sequencer from users requirements and the background », International Computer Music Conference, New York, USA, Juin 2010.