EXÉCUTION RÉPARTIE DE SCÉNARIOS INTERACTIFS

Jean-Michaël Celerier* [1,2], Myriam Desainte-Catherine** [1], Jean-Michel Couturier*** [2]

[1]Univ. Bordeaux, LaBRI, UMR 5800, F-33400 Talence, France
[2] Blue Yeti, F-17110 France

Résumé

La pratique musicale est souvent une opportunité pour interagir et échanger, avant tout avec d'autres musiciens, et plus récemment avec des algorithmes ou logiciels disposant d'un certain degré d'autonomie et de liberté.

La notation musicale occidentale résout le problème du partage et de la séparation d'information entre musiciens en divisant une partition en portées. La plupart des logiciels musicaux vont interpréter ces portées sur une seule machine, en gardant la possibilité de pistes avec des réglages indépendants.

Ce travail consiste en une généralisation de cette notion de partage aux scénarios interactifs, en permettant une exécution synchrone ou asynchrone de différentes parties d'un même scénario sur plusieurs machines, lors d'une seule exécution.

Une implémentation est offerte dans le logiciel i-score, avec pour objectif de mettre en valeur les nouvelles possibilités d'écriture qu'offre une exécution répartie à un compositeur.

1. INTRODUCTION

On cherche à définir une sémantique permettant de décrire l'exécution d'une partition interactive sur plusieurs machines d'un réseau local, en prenant en compte les exécutions parallèles, c'est-à-dire que deux machines jouent ensemble, de manière synchronisée ou non, ainsi que les exécutions sérielles : une machine joue une partie, puis une autre joue la suite. On parle ici de « jeu » de manière abstraite : on s'intéresse au contrôle de tous types de paramètres et non pas uniquement aux paramètres musicaux. Le contexte est celui des partitions interactives telles que définies dans le modèle graphique OSSIA, présenté en [4]. Ses éléments caractéristiques seront brièvement rappelés en section [**sec.rappel-ossia**].

On présentera plusieurs applications et besoins rencontrés par des artistes et auteurs, qui ont motivé ce travail. À partir de ces besoins, on cherche à définir les ajustements nécessaires au modèle OSSIA pour permettre d'exprimer le plus simplement possible des structures réparties dans une partition interactive. Une des premières questions est la nature de la synchronisation désirée entre les machines. Par exemple, des choix d'implémentation sont nécessaire en fonction de la précision désirée. On s'impose de plus pour ce travail un fonctionnement sur du matériel grand public.

Puis, les possibilités de répartition étudiées seront présentées, en analysant l'impact que peuvent avoir les problèmes connus dans le domaine de l'informatique répartie, sur l'écriture de telles partitions. Notamment, on donnera différentes sémantiques possibles pour plusieurs éléments du modèle OSSIA, qui permettent de répondre à des cas d'application distincts. On considérera le partage d'information qu'il peut y avoir entre machines exécutant une même tâche en parallèle, et les différentes manières de synchroniser ces machines entre elles. Pour conclure, les détails d'une première implémentation seront présentés.

2. ÉTUDES DE CAS

2.1. Projet Quarrè

Figure 1 : Quarrè sans participants. Pierre Cochard, 2016.

Quarrè [1] est une installation interactive en son spatialisé réalisée par Pierre Cochard au SCRIME. Elle utilise Max/MSP, i-score, et une application mobile développée pour l'occasion. Elle implique entre un et cinq participants possédant chacun un téléphone.

Une trame principale d'environ une demi-heure se déroule sur une machine fixe. Durant cette trame, à différents moments, les participants vont pouvoir interagir via l'application mobile. Les actions possibles varient en fonction du nombre de participants présents lors de chaque représentation. L'application les avertit par un compte à rebours

* : jcelerier@u-bordeaux.fr
** : myriam@labri.fr
*** : jmc@blueyeti.fr

1 . https://scrime.labri.fr/blog/quarre-composition-interactive

de la date de leur prochaine interaction. Ils disposent ensuite d'une durée déterminée pour agir : changer un paramètre d'un effet, déclencher des sons, spatialiser un objet sonore… Par exemple, entre $t = 1m30$ et $t = 2m$, deux des participants vont pouvoir changer l'intensité d'un effet de chorus, et deux autres vont pouvoir déplacer un objet sonore dans la scène spatiale.

Ce projet permet d'exhiber plusieurs besoins importants :

- La présence d'une forme de polyphonie répartie, avec un nombre de « voix » non fixées au moment de l'écriture.
- L'écriture de cette polyphonie dans le temps : à un instant, au plus deux téléphones sont actifs, à un autre instant aucun ne l'est, etc.
- Le fonctionnement sur des réseaux avec des caractéristiques non idéales, comme le réseau WiFi des téléphones mobiles.

2.2. Besoins additionnels

2.2.1. Support des périphériques embarqués

Actuellement, une des manières principales d'utiliser le logiciel i-score est de lui faire faire du contrôle par réseau.

Néanmoins, pour des applications embarquées, par exemple sur Raspberry Pi, cela peut avoir les conséquences suivantes :

- Encombrement de la bande passante : certains périphériques ne sont pas en Gigabit Ethernet ; envoyer des quantités importantes de messages OSC sur des intervalles de quelques millisecondes va entraîner des pertes de paquets, des retards, et une augmentation de la latence.
- Gigue [2] sur les messages envoyés qui peut entraîner des effets indésirables comme des tremblements si par exemple on fait se déplacer des objets graphiques par réseau.
- Consommation superflue de ressources pour le décodage des messages réseaux.

2.2.2. Lecture de médias synchronisée

On désire offrir des possibilités de lecture permettant à des clips audio ou vidéo de démarrer au même instant sur plusieurs machines, puis de rester synchronisés durant la lecture.

Par exemple, un mur de vidéos synchronisées basé sur de l'embarqué, avec une évolution interactive des vidéos projetées : lorsqu'une personne rentre dans la salle, une vidéo se met à jouer sur tous les écrans ; cette vidéo change à chaque fois qu'une personne sort.

Il est important pour des raisons artistiques que la synchronisation temporelle soit très forte entre les différentes machines : quelques dizaines de millisecondes de décalage entre deux vidéos peuvent déjà être perturbantes pour le public. Le problème est encore plus prononcé avec de l'audio.

2.2.3. Redondance

Le matériel n'étant pas infaillible, la question de la tolérance aux pannes se pose : comment mitiger une panne de l'ordinateur principal pendant une représentation artistique ?

Une des possibilités est d'offrir un système de redondance : si une machine tombe en panne, une autre prend le relais le plus vite possible. La machine prenant le relai doit donc avoir été maintenue en synchronisation avec la machine étant tombé en panne, pour garantir le moins de perturbations possibles lors de la bascule.

3. ÉTAT DE L'ART

3.1. Musique répartie

Il existe plusieurs familles de logiciels et outils offrant un fonctionnement en réseau pour la production musicale. On distingue notamment les applications ayant pour but de synchroniser des flux audio sur plusieurs machines, des applications permettant une collaboration lors de l'écriture et de la composition musicale.

Dans le premier cas, on notera principalement les multiples incarnations du serveur de son NetJack[1], ayant pour but de rendre accessible les éléments d'une baie de raccordement Jack sur plusieurs machines. Ce sont des extensions maître-esclave de Jack qui permettent à une machine d'envoyer un flux audio ou MIDI sur une autre en branchant un câble virtuel. Carôt présente dans [2] un recensement plus complet des serveurs de son adaptés au jeu musical réparti.

Différents logiciels permettant une écriture collaborative, mais non synchronisée, existent : Ohm Studio[7] reprend le principe des séquenceurs traditionnels et permet un partage de document à plusieurs utilisateurs, sur internet. Splice[17] suit le même principe, mais est orienté vers la communauté des *beat-makers*.

Enfin, il existe une famille importante d'outils orientés vers l'improvisation musicale répartie en temps réel[13]. On citera notamment NINJAM [3] et eJamming [4].

3.2. Horloges

En raison de la nature temporelle du problème présenté, on s'intéresse aux mécanismes possibles pour la gestion du temps entre plusieurs machines.

La littérature sur les systèmes distribués distingue plusieurs familles d'horloges :

- Les horloges physiques marquent l'avancement du temps dans le monde matériel.

2 . De l'anglais « *jitter* »

3 . `http://ninjam.com`

4 . `http://www.ejamming.com/`

- Les horloges logiques marquent l'avancement du temps dans les étapes d'un algorithme réparti, donc sans relation avec du temps en secondes.

- Les horloges hybrides ont été récemment introduites afin de concilier ces deux familles.

Les deux principales méthodes pour la synchronisation d'horloges physiques sont *Network Time Protocol*[12] et *Precision Time Protocol*[16]. NTP est disponible sur de nombreuses plate-formes et permet en pratique d'atteindre une précision de synchronisation de quelques milliseconde sur internet. PTP est plus précis et promet une précision proche de la microseconde. Cependant, cela dépend de la précision avec laquelle les paquets sont estampillés [5] et donc de la qualité de l'horloge de l'implémentation PTP. En pratique, l'intérêt de PTP sera plus prononcé lorsque du matériel dédié et onéreux (*Grand Master Clock*) est disponible pour réaliser l'estampillage des paquets.

On rappelle qu'une horloge monotone a pour caractéristique de toujours progresser vers l'avant. Un des désavantages de l'utilisation des horloges physiques pour l'estampillage est qu'elles ne sont pas monotones ; elles peuvent être soumises aux réajustements de l'heure système, et aux secondes interacalaires. Les secondes intercalaires sont des secondes introduites ou supprimées régulièrement [6] dans les systèmes d'heures légaux pour pallier à la divergence entre la rotation terrestre et l'heure UTC obtenue par la mesure de la fréquence de résonance des atomes de césium, qui permet de définir précisément la durée d'une seconde. Leur introduction bouleverse régulièrement les systèmes distribués basés sur les horloges physiques : il est possible que deux appels successifs aux fonctions système donnant l'heure civile renvoient des dates dans l'ordre inverse que celui attendu, ce qui rend les calculs de durées incorrects.

Les horloges logiques ont été introduites par Leslie Lamport [10] afin d'offrir des possibilités de raisonnement et de vérification formelles sur l'écoulement du temps dans les systèmes répartis. En particulier, les estampilles de Lamport permettent d'offrir un ordre partiel entre les messages échangés dans un système réparti.

Ce mécanisme a été par la suite généralisé, d'abord par les horloges vectorielles, puis par les horloges matricielles. Dans le premier cas, chaque message contient un tableau contenant l'horloge de son émetteur ainsi que les horloges connues des autres processus. Dans le second cas, chaque message contient le vecteur contenant les vecteurs d'horloges de chaque processus.

Des solutions existent pour maintenir un lien entre les horloges logiques et physiques. Par exemple, Google a introduit TrueTime dans le cadre de la base de données distribuée Spanner[5]. TrueTime travaille avec des intervalles plutôt que des dates précises, et requiert une synchronisation très précise des horloges physiques, utilisant des signaux GPS et des horloges atomiques.

Les horloges logiques *hybrides*[8] offrent des garanties de causalités sur une horloge physique proche de la précision de NTP, avec une granularité proche de la microseconde.

Enfin, pour les musiciens, des horloges basées non pas sur un temps en secondes, mais sur un temps musical ont été présentées récemment. Elles permettent de s'assurer que plusieurs machines vont jouer sur un même *beat*. C'est le cas par exemple avec Ableton Link [7] . On notera aussi le Métronome Global[14], qui utilise des Raspberry Pi et une connection GPS pour offrir ce genre de synchronisation.

4. APPROCHE

Cette section détaille les choix de haut niveau réalisés.

On souhaite modifier le moins possible le modèle OSSIA, en rajoutant les notions nécessaires et suffisantes pour offrir la finesse de répartition désirée.

La section 5 présente de manière détaillée les possibilités de répartition offertes, en prenant exemple sur des cas simples.

La section 6 définit ces possibilités en utilisant les objets du modèle OSSIA. On notera que cette méthode serait prohibitive à réaliser manuellement par le compositeur : c'est un guide pour réaliser l'implémentation. La répartition se fait automatiquement à partir de la spécification de haut niveau que donne le compositeur via les outils présentés en section 5.

On prendra garde à ne pas confondre ce travail avec la forme de répartition qui existe déjà dans le projet et permet d'orchestrer d'autres logiciels via des protocoles comme OSC.

4.1. Rappel du modèle

On s'intéresse à la problématique de répartition dans le cadre du logiciel i-score, qui implémente le modèle OSSIA [4] pour les partitions interactives. Il consiste en un agencement d'éléments permettant de définir des comportements temporels. Les éléments dont nous avons besoin ici sont :

- **Contrainte temporelle** : décrit un écoulement de temps. Représenté par un trait horizontal. Les contraintes temporelles peuvent être fixes (trait plein) ou souples (trait pointillé), auquel cas leur exécution peut être interrompue par un évènement externe.

- **Nœud temporel** : synchronise le début et la fin de plusieurs contraintes temporelles. Représenté par un trait vertical. Peut porter un **point d'interaction**, qui décrit l'attente et la résolution d'une condition externe pour stopper les contraintes précédentes et démarrer les suivantes ; les contraintes précédentes seront alors en partie représentées par des pointillés pour exprimer l'incertitude sur leur date de fin. Cette incertitude peut être bornée par des dates d'exécution minimales et maximales. La condition prend la

5 . De l'anglais « *timestamped* »

6 . La dernière a été par exemple introduite le 31 décembre 2016 après 23h59 et 59 secondes

7 . `https://www.ableton.com/en/link/`

forme d'une expression logique entre paramètres de la partition. Par exemple,

/couleur == "rouge" && /volume > 1 .

- **Processus** : contenu dans les contraintes temporelles, permet l'exécution d'un comportement à chaque tic d'horloge. Par exemple, une automation.
- **État** : permet d'envoyer un message à un temps donné. Représenté par un point.

Des exemples sont visibles en section 5, par exemple les scénarios 2 et 1.

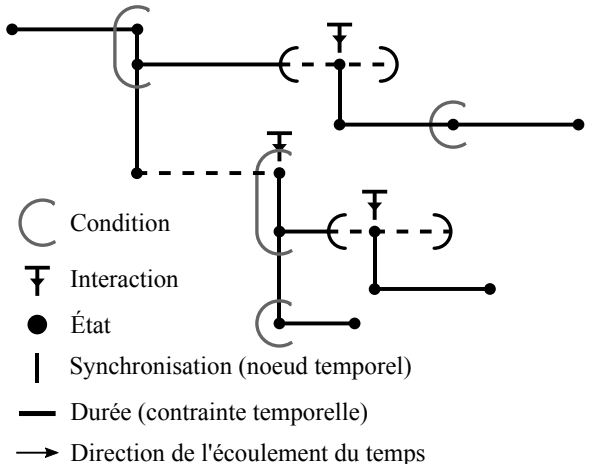


Figure 2 : Éléments du modèle OSSIA.

4.2. Nouvelles notions

Nous nous trouvons en présence de plusieurs ordinateurs qui communiquent et partagent une même structure de donnée : un document. L'ensemble constitué par les instances d'i-score et le document qu'elles partagent est appelé **session**.

On désigne par **client** une instance d'i-score connectée à une session, qui peut être sur une machine différente.

On désire s'affranchir des notions propres aux machines physiques et des problématiques de réseau, telles que adresses IP, ports des machines, lors de l'écriture d'un scénario réparti.

Pour ce faire, la notion de **groupe** est introduite. Un groupe est un ensemble de clients auxquels les mêmes objectifs d'exécution sont assignés. Les compositeurs ne manipulent jamais directement les informations relatives à un client, uniquement celle d'un groupe qui peut contenir zéro, un, ou plusieurs clients.

De manière générale, quand plusieurs clients font partie d'un même groupe, cela signifie qu'ils vont réaliser les mêmes tâches, à des degrés de synchronisation variables.

Un des intérêts de cette approche est la tolérance aux pannes, déconnections, reconnections, et changements d'installation. Par exemple, si une machine tombe en panne, il est possible de la remplacer par une autre simplement en l'assignant au même groupe que la machine en panne sans avoir besoin de mettre à jour le scénario.

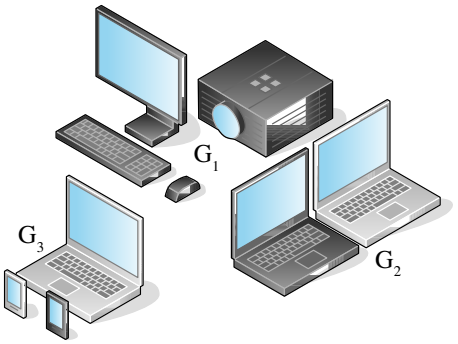


Figure 3 : Plusieurs groupes hétérogènes G_1 , G_2 , G_3.

5. DESCRIPTION DE L'EXÉCUTION

On sépare les objets offrant une structure temporelle, des objets possédant des données et commandes envoyées à d'autres logiciels (par exemple, une automation).

Des groupes et des spécifications de synchronisation sont assignées à ces objets par le compositeur ; en pratique, ces informations sont enregistrées comme une liste de métadonnées associées aux objets.

5.1. Processus de contenu

On entend par processus de contenu, tous les processus produisant des données, sans se soucier de la structure temporelle. En voici une liste non exhaustive : automation, mapping, code javascript, piano-roll MIDI, lecture de son…

Le processus s'exécute tel quel pour tous les clients auquel il est assigné, et ne s'exécute pas pour les autres.

Ici les processus de contenu sont représentés par une boite barrée en diagonale, qui représente le cas courant d'une automation montante.

5.2. Processus Scénario

Le scénario est le processus central d'i-score : il met en relation les différents éléments temporels, en parallèle et en série.

Plusieurs manières de répartir l'exécution d'un scénario, offrant différentes possibilités d'écriture, sont détaillées ci-dessous. On sépare le cas général permettant l'interactivité dans un scénario du cas plus simple où les dates sont fixées. Dans le premier cas (scénario 1), l'avancement dépend du déclenchement d'un évènement externe, tandis que dans le second (scénario 4), l'avancement ne dépend pas de paramètres externes.

On travaillera dans les exemples suivants avec plusieurs groupes $G_{1...N}$ disposant chacun d'un nombre inconnu de clients. On pourra supposer que les contraintes temporelles portent chacune des processus de contenus, qui ne sont pas toujours représentés ici afin de garder les figures simples.

Dans le modèle OSSIA, le nœud temporel est l'objet permettant de synchroniser le début et la fin de plusieurs structures temporelles. Nous analysons d'abord les choix qui doivent être fait pour traduire ce mécanisme dans le cas réparti, puis étudions l'impact sur le processus scénario

dans son ensemble en présentant les politiques de répartition de haut niveau.

5.2.1. Modes de synchronisation

Il est difficile d'offrir une synchronisation forte, par exemple avec une précision d'une milliseconde, dans un système réparti[18]. Une contrainte additionnelle que l'on s'impose est le fonctionnement du système sur du matériel grand public. Ce matériel ne supportera pas toujours des fonctionnalités telles que Ethernet synchrone[6] ou PTP.

On identifie deux possibilités de synchronisation, manifestant leur utilité dans des cas différent :

- Mode synchrone : respecte la sémantique OSSIA. Les éléments s'exécutent dans le même ordre que si le scénario n'était pas réparti, au prix d'une latence augmentée en présence d'interactivité.
- Mode asynchrone : ne respecte pas la sémantique. Une exécution d'un objet peut terminer après que l'exécution de l'objet suivant ait commencé. En revanche, la latence est diminuée.

De plus, on considère la manière dont l'information se propage dans le système :

- Propagation instantanée : lorsqu'une information interactive est disponible dans le système (par exemple « une expression se vérifie »), elle est propagée le plus vite possible aux autres clients qui doivent appliquer le résultat de cette information. Ce mode permet de réduire la latence, au prix de décalages plus importants entre différents clients.
- Propagation compensée : lorsqu'une information est disponible dans le système, elle est propagée de manière à ce que la date absolue de réalisation soit la même (pour un observateur externe) pour tous les clients. On prend en compte pour ce faire les horloges et la latence relative de chaque client ; les dates fixées sont compensées par rapport à cette information. C'est utile notamment dans le cas non-interactif : dès qu'une date peut être fixée, elle l'est, et les clients n'attendent pas de message annonçant la fin. Il convient de rappeler qu'il est physiquement impossible d'exécuter les objets avec la même précision temporelle que s'ils étaient exécutés dans un même tic d'horloge sur le même client ; l'objectif est de minimiser ces décalages temporels.

Les quatre modes possibles permettent de faire des choix à l'écriture en fonction des besoins en terme de consistance et de latence.

Le mode asynchrone instantané va offrir les plus basses latences au détriment de l'ordre d'exécution des objets. Inversement, le mode synchrone compensé permet d'offrir une synchronisation forte qui peut être utile pour des processus média. On aurait par exemple tendance à choisir ce mode pour démarrer des lectures de vidéo synchronisées sur plusieurs machines.

Les différents modes de synchronisation vont impacter :

- L'exécution des points d'interaction.
- La vérification de la validité des conditions.
- Le changement de la vitesse des contraintes temporelles.

Lorsqu'un choix doit être fait, un consensus peut être pris au niveau du groupe auquel est assigné l'objet. Par exemple, la vitesse à laquelle une contrainte temporelle va s'exécuter. Les mécanismes de consensus possibles sont par la suite discutés en section 5.2.5.

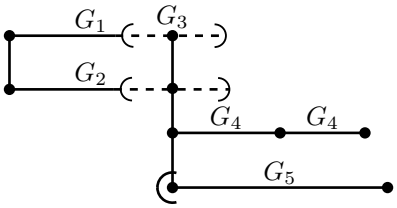
5.2.2. Cas interactif

On dénote trois manières de partager l'information dans un scénario :

- Aucun partage : Les clients associés à ce processus l'exécutent tous de manière indépendante. Les autres ne l'exécutent pas. Si par exemple on assigne le groupe G_1 au scénario 1, tous les clients de G_1 vont exécuter toutes les contraintes temporelles et évaluer toutes les expressions, sans communiquer entre eux sur leurs résultats. Par exemple, pour deux clients de G_1, le point d'interaction pourra se déclencher à des instants différents, et la condition pourra avoir une valeur différente.

  Dans ce cas, les annotations de groupes assignés aux objets du scénario sont récursivement ignorés. Puisque chaque exécution va avoir des temps d'exécution différents pour chaque point d'interaction, il ne peut pas y avoir de synchronisation. La seule politique d'exécution qui pourrait faire sens serait que le premier client à valider un point d'interaction dans un scénario non partagé avertirait les clients suivants.

  Ce cas est utile pour des sous-scénarios dont plusieurs participants à une installation artistique font une expérience partagée, tout en gardant une trame générale de plus haut niveau. Typiquement, on peut imaginer ce cas pour une application mobile.

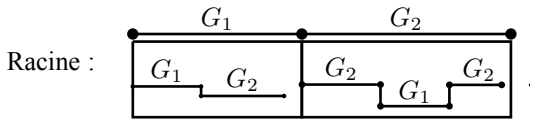


Scénario 1 : Un scénario interactif avec des interactions et des conditions ; $G_{1\ldots5}$ sont les groupes associés aux éléments du modèle. On suppose l'existence de processus de contenus dans chaque contrainte temporelle.

- Partage complet : il n'y a qu'une seule ligne temporelle commune à tous les clients. Les annotations de groupes indiquent l'emplacement d'exécution des processus de contenu et la liste des clients devant parvenir à un consensus pour une expression donnée. Si

par exemple la vitesse d'exécution d'une contrainte est modifiée en temps réel, cette modification est répercutée sur tous les clients.

Cela permet notamment de gérer la répartition d'objets à des niveaux hiérarchiques différents : dans le scénario 2, si le scénario racine est dans ce mode, alors on peut correctement faire exécuter les scénarios enfants en prenant en compte les groupes de leurs objets.



Scénario 2 : Deux contraintes possédant chacune un scénario hiérarchique

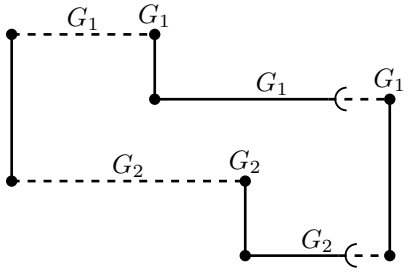
- Mixte : il peut y avoir plusieurs lignes temporelles appartenant à différents groupes dans un même scénario. Ces lignes peuvent ensuite se resynchroniser à un instant donné.

  On considère le graphe dont les nœuds temporels sont les sommets et les contraintes temporelles sont les arêtes. Les lignes temporelles sont les sous-graphes connexes de ce graphe, telles que les contraintes et les nœuds soient associés au même groupe.

  Les annotations donnent l'emplacement d'exécution des contraintes, des processus, et de vérification des expressions.

  Considérons le scénario 3. La différence avec le cas du partage complet tient dans le fait que seuls les clients appartenant au groupe G_1 vont exécuter la branche du haut. Ils peuvent ou non avoir l'obligation de se synchroniser entre eux. Il faut alors qu'une synchronisation ait lieu pour tous les clients appartenant à G_1 et G_2 lors du dernier point d'interaction, situé à droite.

  Comme pour le premier cas, en raison de possibilités d'exécutions divergentes du même contenu, il est impossible d'offrir une répartition hiérarchique cohérente.



Scénario 3 : Deux branches exécutées chacunes par un groupe différent.

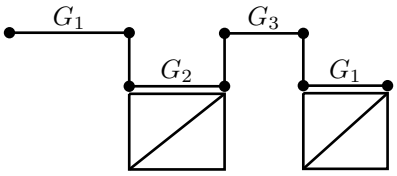
5.2.3. Cas non-interactif

On peut utiliser les méthodes de synchronisation décrites précédemment, mais il est possible d'étendre le mode compensé.

Comme on connait les dates effectives auxquelles les objets sont sensés s'exécuter, il est possible de les fixer à l'avance sur chaque client. Un parcours de graphe permet d'obtenir une estimation des dates minimales auxquelles il sera possible de fixer des dates d'exécutions des éléments suivant un point d'interaction donné. Cette méthode est notamment développée dans [3].

Ce principe de pré-calcul permet d'offrir dans une certaine mesure une tolérance au partitionnement : s'il y a une déconnexion, l'exécution va continuer à fonctionner au moins jusqu'au prochain point d'interaction. Cela peut laisser du temps à un régisseur pour régler le problème. En revanche, puisqu'il n'y a pas d'ordonnancement au moment des points d'interaction, il suffit d'un peu de délai pour que, dans le scénario 4, le début de l'exécution sur G_2 survienne avant la fin de l'exécution sur G_1. Il est donc particulièrement important dans ce cas de garder les horloges des machines synchronisées.

On notera que modifier la vitesse d'exécution des éléments en temps réel implique de mettre à jour les estimations de dates sur tous les autres clients. En autorisant cela, on perd donc aussi cette tolérance au partitionnement et la cohérence de l'exécution pourra être brisée.



Scénario 4 : Les groupes G_1, G_2, G_3 exécutent des contraintes temporelles pouvant contenir des processus les uns à la suite des autres

5.2.4. Expressions et interactivité

Dans le cas sans partage, où un scénario est exécuté intégralement en parallèle par différents clients, il n'y a pas de situation particulière à gérer. Chaque client vérifie les expressions en fonction des données dont elle dispose, et les valide à l'instant ou elle le souhaite. Cela peut être utile si par exemple la partition implique plusieurs téléphones qui font tous tourner un scénario semblable, mais où chaque individu peut choisir de faire avancer le scénario au rythme où il le souhaite.

Dans le cas d'un partage de certaines lignes temporelles par plusieurs clients, il est nécessaire de synchroniser des informations partagées. Ce sont par exemple les expressions, leurs dates d'évaluation, et la vitesse d'exécution des contraintes temporelles.

- S'il n'y a qu'une seule ligne temporelle, chaque expression doit n'avoir qu'une valeur de vérité. Il doit

donc y avoir un consensus sur la valeur de cette expression. De même pour la vitesse d'exécution des contraintes temporelles.

Plusieurs manières d'obtenir un consensus sont possibles :

- Dans le cas des conditions, au moins un client valide ou nie l'expression.
- Dans le cas des points d'interaction, on peut fixer la valeur de l'expression à celle du premier client qui la vérifie, à l'aide des estampilles.
- Dans le cas des points d'interaction, une majorité de clients valident l'expression.
- Tous les clients valident ou nient l'expression.

- Dans le cas mixte, la synchronisation doit alors uniquement se faire à l'entrée ou à la sortie d'une branche.

On note plusieurs problèmes possibles :

- La déconnexion d'un client lors d'une prise de décision. Dans ce cas, on peut s'informer de la déconnexion, en effectuant un ping régulier, et prendre la décision avec les participants restants.
- Le cas d'un *ex æquo*, si un groupe a un nombre pair de participants. Il existe plusieurs possibilités de résolution :
  - Choisir en fonction des estampilles : le premier choix est celui qui est retenu.
  - Nommer un chef de groupe qui permet de départager.
  - Choisir au hasard.

5.2.5. Consensus

Comme mentionné précédemment, dans plusieurs cas, différents clients doivent s'accorder sur le résultat d'une expression.

Paxos et ses variantes[9], ainsi que Raft[15] sont des algorithmes permettant de garantir un consensus sur une information dans un système réparti.

On sépare la synchronisation du consensus du mécanisme d'exécution qui suit la résolution de ce consensus :

1. Les nœuds impliqués dans l'expression décident de la valeur de vérité.
2. Une fois cette valeur connue, tous les nœuds précédant, suivant, et impliquant l'expression sont inclus dans la décision de la date d'exécution.

5.3. Alternative : répartition bas niveau

Une approche possible mais non considérée dans ce travail serait de réaliser la synchronisation à chaque tic d'horloge, via un mécanisme d'asservissement maître-esclave.

Le maître s'exécute avec une horloge physique. À chaque tic, il envoie à tous les clients la date à laquelle ils doivent exécuter leur propre tic. Quand les clients reçoivent ce message, ils appliquent une attente active [8] jusqu'à ce que l'exécution du tic prenne place.

5.4. Récapitulatif

On introduit une notion de client (potentiellement une machine physique) et de groupe. Un groupe peut contenir plusieurs clients, et un client peut être présent dans plusieurs groupes.

On choisit pour la répartition un modèle de document partagé : tous les clients voient le même document, mais peuvent chacun en interpréter des sous-parties différemment, à la discrétion du compositeur.

Différents objets du modèle d'i-score peuvent être assignés à un ou plusieurs groupes :

- Les processus
- Les points d'interaction
- Les contraintes temporelles

Dans le cas des points d'interaction, plusieurs niveaux de synchronisation sont présentés.

Donner un groupe à une contrainte temporelle propage ce groupe récursivement à tous ses processus enfants.

Pour le cas du processus scénario, la répartition peut avoir lieu avec divers degrés de partage d'information entre les clients.

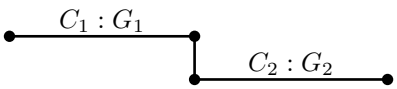
6. SÉMANTIQUE

On décrit ici la sémantique des différentes méthodes de synchronisation vues précédemment.

On fait le choix de décrire ce modèle via les primitives de OSSIA. En effet, on dispose déjà par ce biais de méthodes de synchronisation et d'envoi de message.

Pour chaque scénario tel qu'écrit par le compositeur, comme les scénarios 5, 6, 7, on donne la transformation requise pour passer à un scénario réparti.

6.1. Déplacement sans interactivité



Scénario 5 : Deux contraintes temporelles se suivent sur deux groupes différents

Le premier cas, visible dans le scénario 5 est celui du déplacement. Deux contraintes C_1, C_2 se suivent. On note $C_1 : G_1$ le fait que la contrainte temporelle C_1 s'exécute sur les machines du groupe G_1. Le basculement de C_1 à

8 . De l'anglais « *busy wait* » : le programme boucle jusqu'à ce qu'une condition de déclenchement soit validée.

C_2 se fait à une date connue à l'avance : on n'attend pas d'interactions.

On applique les transformations suivantes :

1. On crée une bifurcation au niveau du premier nœud temporel, avant C_1.
2. On introduit une contrainte temporelle $C_{1\to 2}$.
3. On déplace C_2 à la suite de cette contrainte temporelle.

4a. Cas instantané asynchrone (fig. 4a).

On crée un message M_1 à la fin de C_1 qui ira déclencher automatiquement le point d'interaction T_1. Cela implique la souplesse de $C_{1\to 2}$. Ce message sera envoyé lorsqu'un consensus sur le groupe G_1 sera établi, avec une des politiques énoncées en section 5.2.4.

4b. Cas compensé synchrone (fig. 4a).

Lorsqu'un consensus est établi sur la date de fin de C_1, un message M_1 est envoyé aux clients de G_2 avec cette date, de manière à ce que leur exécution démarre en même temps que la fin de C_1.

4c. Cas compensé asynchrone (fig. 4b).

Le message M_1 est envoyé au début de C_1 plutôt qu'à la fin. Sur les machines de G_1, le point d'interaction T_1 n'attend rien. Il attend M_1 sur les machines de G_2. Les clients du groupe G_1 et G_2 peuvent être déconnectés après le début de la lecture et continueront à fonctionner comme prévu.

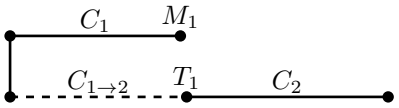
On remarquera trois choses :

- Puisqu'il n'y a pas d'expressions, le mode instantané synchrone n'apporte rien par rapport au mode instantané asynchrone. En effet, le démarrage de C_2 implique dans tous les cas la fin de C_1, car ce sont les clients du groupe qui évaluent C_1 qui décident de la date d'exécution du nœud temporel les séparant, contrairement au cas ou une expression serait évaluée par un autre groupe que G_1 ou G_2.
- Le cas compensé synchrone correspond à l'exécution hypothétique ou chaque machine dispose d'horloges parfaites.
- Le cas compensé avec pré-calcul est incompatible avec une synchronisation forte : on suppose que la synchronisation se fera automatiquement via la synchronisation des horloges sous-jacentes.

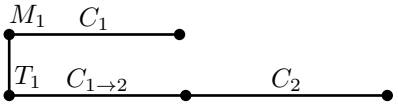
6.2. Déplacement avec interactivité : cas simple

On étudie d'abord un cas simple similaire au cas non-interactif, mais avec une attente d'une condition externe pour le déclenchement.

Le scénario 6 peut se résoudre de plusieurs manières.

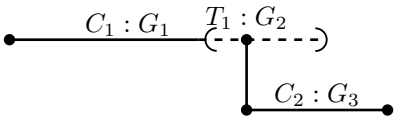


(a) Cas instantané et compensé synchrone



(b) Cas compensé asynchrone, avec pré-calcul

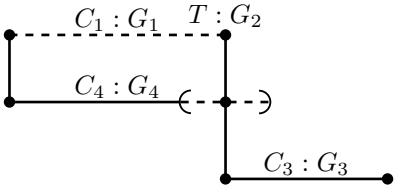
Figure 4 : Transformations pour l'exécution de deux contraintes en série sur deux clients séparées



Scénario 6 : Des groupes G_1 et G_3 exécutent les deux contraintes temporelles tandis qu'un groupe G_2 atteint un consensus sur l'expression

- Si l'exécution du nœud est effectuée de manière instantanée, tous les clients de G_2 envoient l'information de déclenchement à tous les clients de G_1 et G_3, sans garantie d'ordre. Les points d'interaction se déclenchent immédiatement.
- Si l'exécution du nœud est effectuée de manière compensée :
  - Quand la condition devient vraie, une date d'arrêt des clients de G_1 est fixée dans le cas synchrone, et des clients de G_1 et G_3 dans le cas asynchrone. De plus, dans le cas synchrone, on applique un mécanisme de consensus similaire permettant de garantir que tous les membres de G_1 ont terminé leur exécution avant le démarrage de la contrainte assignée à G_3.
  - Si on atteint la borne maximale : un client est nécessairement le premier à atteindre cette borne. Il peut alors en informer les autres, qui vont déclencher le passage à la suite.

6.3. Déplacement avec interactivité : cas étendu



Scénario 7 : Un scénario possédant un point d'interaction que l'on veut répartir sur les clients des groupes $G_{1,..,4}$

On considère ici le scénario 7, qui possède un point d'interaction T, géré par le groupe G_2.

On note :

- T_e : Le point d'interaction qui marque le début de l'évalutation de l'expression de T.
- T_2 : Le point d'interaction marquant le consensus sur l'expression de T.
- T_1 : Le point d'interaction qui marque la fin de l'exécution de la contrainte de G_1. Est déclenché par M_1.
- T_4 : Le point d'interaction qui marque la fin de l'exécution de la contrainte de G_4. Est déclenché par M_4.
- T_3 : Le point d'interaction qui marque le début de l'exécution de la contrainte de G_3. Est déclenché par M_3.

Dans le cas asynchrone, en scénario 8, M_1 , M_3 , M_4 sont des messages qui vont déclencher les points d'interaction T_1 , T_3 , T_4 dès qu'ils sont reçus.

Dans le cas synchrone, en scénario 9, les messages partent le plus tôt possible, mais le déclenchement de T_3 implique la fin de l'exécution de C_1 et C_4. T_3 démarrera lorsque tous les messages M_3 auront été envoyés : son expression est une conjonction logique sur la réception de $M_{3_{1,..N}}$ avec 1 , . .N les groupes précédant T.

Dans les cas compensés, la structure ne change pas. Les messages envoyés changent : M_1 , M_3 , M_4 fixent la date à laquelle T_1, T_3, T_4 doivent partir de manière à ce qu'ils se déroulent en même temps.

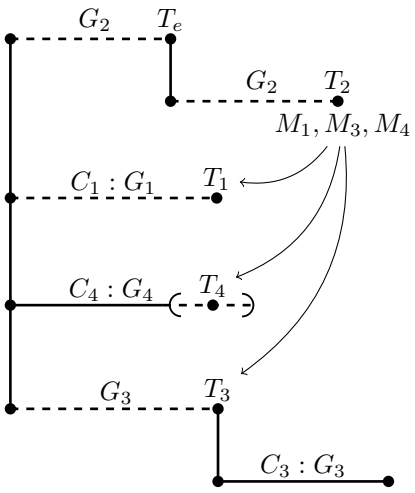
Il serait possible d'exprimer des variantes plus fines : par exemple rendre simultanées les fins de C_1 , C_4 mais pas le début de C_3. Néanmoins, ce travail n'a pas pour objectif d'encombrer le compositeur d'une finesse trop rarement nécessaire, en lui offrant un accès facile à des concepts de haut-niveau dont la nécessité est apparue lors des études de cas.

7. IMPLÉMENTATION

Le système présenté en section précédente est agnostique à la topologie sous-jacente du réseau : on définit les messages qui doivent être envoyés et reçus, mais pas la manière dont ils transitent.

La mise en œuvre, réalisée en C++ en tant que plug-in à i-score, est dite « de référence » : l'objectif est de valider l'approche, sans chercher les performances maximales. Pour des raisons de simplicité, on choisit une topologie réseau en étoile : un maître gère le déroulement général de l'exécution. Les différents clients communiquent au travers de ce maître pour les messages de balisage, avec les compromis que cela implique :

- Facilité d'analyse et de débogage lors du développement : on peut enregistrer tous les messages échangés avec leurs estampilles.



Scénario 8 : Répartition du scénario 7 dans le cas asynchrone instantané : les contraintes associées aux groupes A , D et la contrainte associée au groupe G_3 s'arrêteront et démarreront sans ordre prévisible.

- Simplicité du consensus : le maître prend la décision avec les informations envoyées par les clients, puis les informe.
- Intolérance aux pannes : si le maître faillit, aucune récupération n'est possible.
- Latence non minimale : plus de messages sont échangés que si tous les clients communiquaient directement.

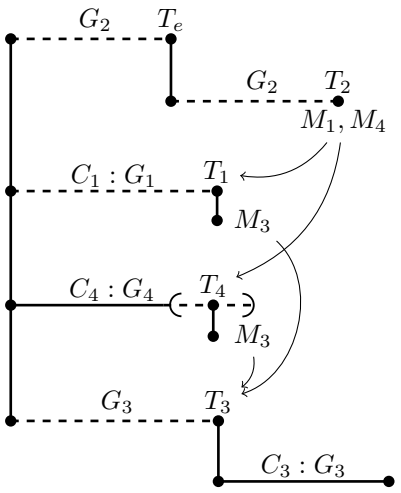
D'autres implémentations en mode pair-à-pair seraient possibles.

On utilise dans l'implémentation le protocole TCP pour tous les messages relatifs à la répartition, notamment pour ses garanties d'ordonnancement. En effet, il a été démontré qu'UDP seul était généralement peu fiable pour des messages de contrôle. Des architectures plus complexes telles que le « Modèle Client-Serveur Généralisé » basées elles mêmes sur UDP peuvent néanmoins compenser l'absence de fiabilité[11], tout en offrant de meilleures performances que TCP.

Enfin, de manière générale, on repose sur une bonne synchronisation des horloges au niveau du système. L'algorithme d'exécution d'i-score compense en effet les délais à chaque tic si l'horloge système se resynchronise. Plus la synchronisation des horloges sera fine entre les clients et plus l'exécution se rapprochera de l'exécution théorique non répartie.

8. CONCLUSION

On présente une méthode pour répartir l'exécution de parties d'un scénario interactif basé sur le modèle OSSIA sur différentes machines d'un réseau local. L'objectif est d'offrir de nouvelles possibilités d'écriture au compositeur travaillant dans un environnement hétérogène, et compor-



Scénario 9 : Répartition du scénario 7 dans le cas synchrone instantané : les contraintes associées aux groupes G_1 , G_4 et la contrainte associée au groupe G_3 s'arrêteront dans l'ordre, avec un délai potentiel entre la fin de G_1 , G_4 et le début de G_3.

tant téléphones, cartes embarquées comme Raspberry Pi, et machines de bureau beaucoup plus puissantes.

Les différentes possibilités de synchronisation des structures temporelles sont introduites. D'abord au niveau d'un nœud temporel, en permettant divers degrés de cohésion avec la structure donnée par le compositeur, allant d'exécutions pas nécessairement simultanées mais avec une latence faible, à des exécutions avec une latence plus importante, mais permettant une plus grande simultanéité pour l'observateur. Puis au niveau d'un scénario, où l'on considère les possibilités de partage qu'il peut y avoir entre plusieurs machines exécutant un même scénario : toutes en parallèle, certaines en série, etc.

La suite de ce travail consistera à étendre les possibilités d'écriture pour inclure des informations d'interaction plus complexes. Par exemple, on souhaiterait pouvoir contrôler un paramètre tel que le volume d'un son en temps réel, en prenant en compte les interactions de plusieurs participants sur leur téléphone. Des comportements tels que « prendre la valeur moyenne », « prendre la valeur la plus élevée » pourraient alors être introduits dans la palette à disposition de l'auteur.

9. REMERCIEMENTS

Les auteurs souhaitent remercier Serge Chaumette et Simon Archipoff pour leurs contributions.

Références

[1] Alexander Carôt, Torben Hohn et Christian Werner. « Netjack-Remote music collaboration with electronic sequencers on the Internet ». In : *Linux Audio Conference*. 2009.

[2] Alexander Carôt, Pedro Rebelo et Alain Renaud. « Networked music performance : State of the art ». In : *Audio engineering society conference : 30th international conference : intelligent audio environments*. Audio Engineering Society. 2007.

[3] Jean-Michaël Celerier, Myriam Desainte-Catherine et Jean-Michel Couturier. « Rethinking the audio workstation : tree-based sequencing with i-score and the LibAudioStream ». In : *Sound and Music Computing*. 2016.

[4] Jean-Michaël Celerier et al. « OSSIA : Towards a unified interface for scoring time and interaction ». In : *TENOR2015*. 2015.

[5] James C Corbett et al. « Spanner : Google's globally distributed database ». In : *ACM Transactions on Computer Systems (TOCS)* 31.3 (2013), p. 8.

[6] J-L Ferrant et al. « Synchronous Ethernet : A method to transport synchronization ». In : *IEEE Communications Magazine* 46.9 (2008).

[7] Martin K Koszolko. « Crowdsourcing, jamming and remixing : a qualitative study of contemporary music production practices in the cloud ». In : *Journal on the Art of Record Production* 10 (2015).

[8] Sandeep S Kulkarni et al. « Logical physical clocks ». In : *International Conference on Principles of Distributed Systems*. Springer. 2014, p. 17–32.

[9] Leslie Lamport. « The part-time parliament ». In : *ACM Transactions on Computer Systems (TOCS)* 16.2 (1998), p. 133–169.

[10] Leslie Lamport. « Time, clocks, and the ordering of events in a distributed system ». In : *Communications of the ACM* 21.7 (1978), p. 558–565.

[11] Curtis McKinney et Chad McKinney. « Oscthulhu : Applying video game state-based synchronization to network computer music ». In : *International Computer Music Conference*. 2012.

[12] David L Mills. « Internet time synchronization : the network time protocol ». In : *Communications, IEEE Transactions on* 39.10 (1991), p. 1482–1493.

[13] RH Mills. « Dislocated sound : A survey of improvisation in networked audio platforms ». In : *New Interfaces for Musical Expression*. University of Technology, Sydney. 2010.

[14] Reid Oda, Rebecca Fiebrink et al. « The Global Metronome : Absolute tempo sync for networked musical performance ». In : *New Interfaces for Musical Expression*. Goldsmiths University of London, 2016.

[15] Diego Ongaro et John Ousterhout. « In search of an understandable consensus algorithm ». In : *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. 2014, p. 305–319.

[16] Yu Peng-Fei et al. « The research of precision time protocol IEEE1588 ». In : *International Conference on Electrical Engineering*. 2009.

[17] Joseph Michael Pignato et Grace M Begany. « Deterritorialized, Multilocated and Distributed : Musical space, poietic domains and cognition in distance collaboration ». In : *Journal of Music, Technology & Education* 8.2 (2015), p. 111–128.

[18] Justin Sheehy. « There is no now ». In : *Communications of the ACM* 58.5 (2015), p. 36–41.