

ÉCRITURE ET EXÉCUTION RÉPARTIE DE SCÉNARIOS INTERACTIFS

Auteur 1
Organisme
Adresse électronique

Auteur 2
Organisme
Adresse électronique

Auteur 3
Organisme
Adresse électronique

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 plusieurs parties d'un scénario sur plusieurs machines. 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. PROBLÉMATIQUE

On cherche à définir une sémantique permettant de décrire l'exécution d'une partition interactive sur plusieurs machines, en prenant en compte les exécutions parallèles, c'est-à-dire que deux machines jouent la même chose, de manière synchronisée ou non, ainsi que les exécutions série : une machine joue puis une autre machine joue la suite.

On parle ici de « jeu » à un niveau abstrait : on s'intéresse au contrôle de tous types de paramètres et non pas uniquement les paramètres musicaux.

On présentera d'abord plusieurs applications et besoins rencontrés par des artistes et auteurs, qui ont motivé ce travail. Puis, les possibilités de répartition étudiées seront présentées dans le détail, en analysant l'impact que peuvent avoir les problèmes connus dans le domaine de l'informatique répartie, sur l'écriture de telles partitions. Pour conclure, les performances du système développé seront présentées.

2. ÉTUDES DE CAS

2.1. Projet Quarrè

Quarrè (fig. 1) est une installation en son spatialisée réalisée par Pierre Cochard au SCRIME, qui utilise

Max/MSP, i-score, et une application mobile développée pour l'occasion. Elle implique plusieurs participants possédant chacun un téléphone.

Une trame principale d'environ trente minutes se déroule. Durant cette trame, à différents moments, des participants vont pouvoir interagir via l'application mobile, qui les avertis par un compte à rebours. Ils disposent ensuite d'une durée déterminée pour agir : changer un paramètre d'un effet, déclencher des sons, spatialiser un objet sonore... Les actions possibles varient en fonction du nombre de participants, pouvant aller de un à cinq.



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

2.2. 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, sur des périphériques embarqués, cela a un coût :

- 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 l'augmentation du temps de latence.
- Gigue ¹ 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 de ressources pour le décodage des messages réseaux.

1. De l'anglais jitter

De plus, actuellement, il est nécessaire de copier les fichiers de sauvegarde sur le périphérique à chaque modification que l'on veut tester, ce qui est peu ergonomique. On essaye donc par la même occasion d'offrir un système qui permette de partager automatiquement un scénario i-score entre deux machines.

2.3. Lecture de médias synchronisée

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

2.4. Redondance

Le matériel n'étant pas infallible, 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 relais doit donc avoir été maintenue en synchronisation avec la machine étant tombée en panne, pour garantir le moins de perturbations possibles lors de la bascule.

2.5. État de l'art

2.5.1. Rappel du modèle d'i-score

Ce document présente la problématique de répartition dans le cadre du logiciel i-score, qui implémente le modèle OSSIA[3]. 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.
- **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.
- **Processus** : contenu dans les contraintes temporelles, permet l'exécution d'un comportement à chaque tic d'horloge. Par exemple, une automatisation.
- **État** : permet d'envoyer un message à un temps donné. Représenté par un point.

Des exemples sont visibles en section 4, par exemple les scénarios 1 et 2.

2.5.2. Répartition à l'exécution

2.5.3. Horloges

En raison de la nature temporelle du problème présenté, on s'intéresse avant tout 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.
- Les horloges logiques marquent l'avancement du temps dans les étapes d'un algorithme réparti, donc sans relation avec du temps en secondes.
- Plus récemment, les horloges hybrides ont été introduites afin de concilier les deux mondes.

Les deux principaux mécanismes pour la synchronisation d'horloges physiques sont NTP et PTP.

Les horloges logiques ont été introduites par Leslie Lamport [4] afin d'offrir des possibilités de vérification sur les systèmes répartis.

Problème de l'horloge :

- Si on utilise l'horloge système, pas de garantie qu'elle soit bien synchronisée. Et NTP pas disponible partout (on n'a pas forcément les droits sur un téléphone pour changer l'heure). Horloge i-score se resynchronise déjà en cas de délai sur une machine.
- Si on utilise une horloge interne, problème de la synchronisation avec l'horloge système (pour le tick)

<http://www.cse.buffalo.edu/tech-reports/2014-04.pdf>
<http://queue.acm.org/detail.cfm?id=2745385> <http://radar.oreilly.com/2012/10/google-spanner-relational-database.html> <http://www.ntp.org/ntpfaq/NTP-s-sw-clocks-quality.htm>

Faire la relation entre notre système et le problème CAP : https://en.wikipedia.org/wiki/PACELC_theorem

Horloges logiques : équivalent à envoyer une timestamp à chaque tick (on arrive donc à un mécanisme de vector clock approximatif).

Synchro d'horloges de deux clients dont une qui ralentit, si elles exécutent le même scénario ? Il faut les recalculer régulièrement.

Possibilité : externe (NTP, etc) ou interne : <https://github.com/ethanlim/NetworkTimeProtocol>

Recaler dès qu'on accumule du retard ?

On peut introduire un recalage sur la master clock entre chaque tick.

Ce qu'on fait : ping régulier vers chaque client (toutes les 100 millisecondes)

Quand quelque chose doit se synchroniser, on dit à chaque client à quel instant il est supposé arriver par rapport à son horloge système.

Quand un client reçoit un ordre pour un timenode à t , il l'applique dès que $t \leq \text{local}(t)$ (modulo un tick ?)

Possibilité : synchronisation via démon externe (PTP, NTP...), mais pas toujours possible (on ne peut pas supposer que l'utilisateur a les droits pour changer l'horloge

sur sa machine).

Synchronous ethernet

Ableton Link : synchro sur les ticks musicaux

Avoir une horloge propre à i-score ? Mais du coup maintenant il faut la synchroniser à l'horloge système.

2.5.4. Problématique de la sécurité

On veut : résistance aux "normal" failures, pas Byzantine failures.

2.5.5. Synchronisation de médias

- pas pour l'instant... doivent être au même endroit sur la même machine. Ableton Link ? Netjack ?

3. APPROCHE

Cette section détaille les choix de haut niveau réalisés.

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

La section 4 présente de manière détaillée les possibilités de répartition que l'on offre, en prenant exemple sur des cas simples.

La section 5 définit ces possibilités en utilisant les objets du modèle d'i-score. On notera que cette méthode serait prohibitive à réaliser manuellement : c'est un modèle pour réaliser l'implémentation, qui se fait automatiquement à partir de la spécification que donne le compositeur via les outils présentés en section 4. On prendra donc garde à ne pas confondre la forme de répartition qui existe déjà de fait dans le logiciel, l'objectif premier d'i-score étant de communiquer avec d'autres logiciels, de la répartition entre instances d'i-score partageant un même document.

Enfin, la section 6 présente une évaluation des performances dans le cadre d'une implémentation pratique.

----- Première approche basées sur réseau de Pe-tri.

Puis migration du modèle d'i-score maintenant autonome.

- Problème des devices. Note : si on se permet d'envoyer des messages OSC quelconques (i.e. pas dans l'arbre), ça peut simplifier des choses. Aussi, permettre devices qui sont juste en mode envoi ? Il faut un autre proto que OSC ou Minuit...

Aussi, refactoriser pour utiliser tout i-score en librairie

Différentes possibilités de répartition sont disponibles selon les types de processus.

Détailler les implications des choix par rapport au théorème CAP et PACELT.

3.1. Nouvelles notions

Nous nous trouvons en présence de plusieurs machines qui communiquent et partagent 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.

On désire s'affranchir des notions propres aux machines physiques et des problématiques de réseau (adresse IP, etc) lors de l'écriture d'un scénario réparti.

Pour ce faire, on introduit la notion de *groupe*. Un groupe est un ensemble de clients.

Les compositeurs ne manipulent jamais directement la notion de 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.

L'intérêt d'un groupe est la tolérance aux pannes, déconnexions, reconnexions, et changements d'installation. Par exemple, si une machine tombe en panne, on peut 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.

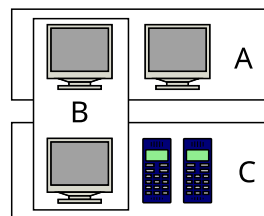


Figure 2 : Plusieurs groupes A, B, C avec plusieurs clients dans chaque groupe

3.2. Topologie

Maître-esclave pour l'édition.

Pour des raisons de simplicité, la première mise en œuvre se fait via une topologie réseau en étoile : un maître gère le déroulement général de l'exécution. Les différents clients communiquent au travers de ce maître, avec les avantages et inconvénients que cela implique :

- Facilité d'analyse lors du développement : on peut enregistrer tous les messages échangés avec leurs estampilles.
- Intolérance aux pannes : si le maître faillit, aucune récupération n'est possible.

On détaille dans les sections suivantes les cas où des topologies différentes peuvent montrer un intérêt.

Pour l'exécution : dans certains cas deux clients peuvent échanger directement. Il faut décrire ces cas.

Si groupes avec plusieurs clients : "group leader" ? responsable de la communication ? avoir un "plus court chemin" permanent ?

3.3. Modèle

i-score se base sur une approche orientée document.

On choisit de répartir nos objets autour d'un même document partagé par toutes les instances du réseau, à la manière des logiciels d'édition tels que Google Docs.

Cela ne pose pas de problème de mémoire, même sur de l'embarqué tel que Raspberry Pi : l'empreinte mémoire d'un scénario est minime en comparaison de l'occupation des bibliothèques utilisées (quelques dizaines de kilo-octets contre environ 50 méga-octets).

On utilise le protocole TCP pour tous les messages relatifs à la répartition, notamment pour ses garanties d'ordonnancement.

3.4. Répartition de l'édition

- Partage de file undo - redo. Transformations opérationnelles.
- Autres approches : partage du modèle lui-même.
- Pour l'édition temps-réel, on doit permettre d'appliquer des "filtres" (par exemple qui vont rajouter une sur-expression, etc)

4. 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 automatisation).

Des groupes et des méthodes de synchronisation sont assignées à ces objets par le compositeur ; en pratique, ces informations sont enregistrées comme une liste de méta-données associées aux objets. Il est possible d'assigner plusieurs groupes à un objet ; cela se comporte comme un groupe constitué de la somme des clients de chaque groupe était assigné.

4.1. Processus de contenu

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

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 boîte barrée en diagonale, qui représente le cas courant d'une automation montante.

4.1.1. Comportements de groupes

-> pourquoi ne pas avoir la même chose au niveau des mappings (et pour tout ce qui n'est pas fixé de manière générale ?) Deux manières : offrir un processus de mapping spécialisé qui permet de définir les relations qu'on veut dans un groupe ("first", "all", "mean", "sum", etc...)

Ou bien faire cette transformation à plus bas niveau (pour les devices) : les sorties sont constantes mais on peut appliquer des transformations aux entrées. Par

exemple, si on possède un device spécial "network" avec l'arbre local de chaque "groupe", qui offre une couche d'abstraction au-dessus des mécanismes de consensus, on peut avoir des possibilités plus riches, telles que : l'expression e_2 est vraie si l'expression e_1 était vraie pour le groupe B et fausse pour le groupe A .

Pour permettre plus de possibilités, on introduit la possibilité de réaliser des expressions qui réalisent un pattern-matching : `network : /*/ trigger == true`.

4.2. Processus Scénario

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

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 (le scénario continue après qu'un événement externe e se soit déclenché, visible par exemple en scénario 2) du cas plus simple où les dates sont fixées, visible en scénario 4.

On travaillera dans les exemples suivants avec trois groupes A, B, C 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 représentés ici afin de garder les figures simples.

4.2.1. Modes de synchronisation

Il est difficile d'offrir une synchronisation forte (par exemple avec une précision d'une milliseconde, dans un système réparti[5]. Une contrainte additionnelle est le fonctionnement du système sur du matériel non-professionnel, n'offrant donc pas le support de fonctionnalités telles que Ethernet synchrone ou PTP.

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

- Asynchrone : lorsqu'une information interactive est disponible dans le système (par exemple «une expression se vérifie»), elle est propagée le plus vite possible aux autres nœuds qui doivent appliquer le résultat de cette information. Ce mode ne respecte pas nécessairement la sémantique avant-après de i-score mais permet de réduire la latence.
- Simultané : 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. 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 les décalages temporels.

De plus, on considère les possibilités d'ordonnement des événements lors de la communication entre

machine :

- Mode ordonné : respecte la sémantique d'i-score : 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 non-ordonné : ne respecte pas la sémantique d'i-score : une exécution d'un objet peut terminer après que l'exécution de l'objet suivant ait commencé. En revanche, la latence est diminuée.

De manière générale, on repose sur une bonne synchronisation des horloges au niveau du système. i-score est capable de compenser les délais 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 dans le cas non réparti.

Les différents modes de synchronisation vont impacter :

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

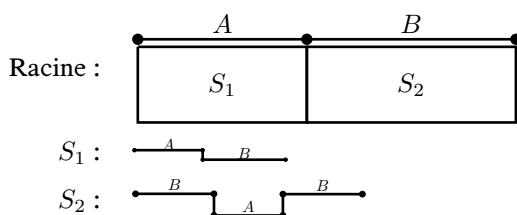
Lorsqu'un choix doit être fait, un consensus peut être pris au niveau du groupe auquel est assigné l'objet. Par exemple, quelle va être la vitesse à laquelle une contrainte temporelle va s'exécuter. Les mécanismes de consensus possibles sont détaillés plus en détail par la suite.

4.2.2. Cas interactif

Trois niveaux de partage :

- Partage complet : il n'y a qu'une seule ligne temporelle partagée pour tous les clients. Les annotations de groupes indiquent l'emplacement d'exécution. Elles servent à indiquer ou non l'exécution d'un processus sur un client, et le groupe 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 qui l'exécutent.

Cela permet notamment de gérer la répartition d'objets à des niveaux hiérarchiques différents : dans le scénario 1, 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.

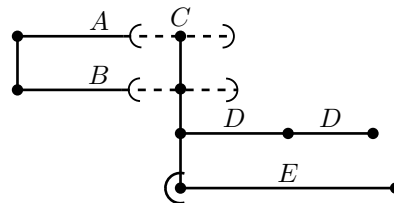


Scénario 1 : Deux contraintes possédant chacune un scénario hiérarchique

- Aucun partage : les clients n'étant pas associées à ce processus ne l'exécutent pas, ceux qui y sont associées l'exécutent tous de manière indépendante. Si par exemple on assigne le groupe *A* au scénario 2, tous les clients de *A* vont exécuter tous les éléments, sans communiquer sur leurs résultats. Par exemple, pour deux clients *A*₁ et *A*₂, le point d'interaction pourra se déclencher à des instants différents, et la condition pourra avoir une valeur différente.

Cela implique que les annotations de groupes assignés aux objets du scénario sont ignorés, récursivement : puisque chaque exécution va avoir des temps différents par conception, il ne peut pas vraiment 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 notamment utile pour avoir des sous-scénarios dont plusieurs participants à une installation artistique peuvent faire l'expérience en même temps, tout en gardant une trame générale de plus haut niveau. Typiquement, on peut imaginer ce cas pour une application mobile.

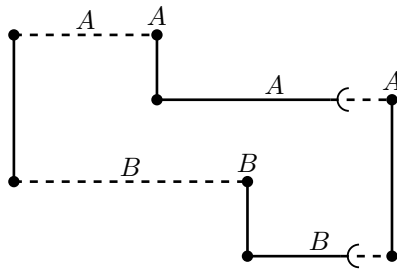


Scénario 2 : Un scénario interactif avec des interactions et des conditions ; *A*, *B*, *C*, *D*, *E* sont les groupes associés aux éléments du modèle. On suppose l'existence de processus de contenus dans chaque contrainte temporelle.

- 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.

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 «partage complet» tient dans le fait que seuls les clients appartenant au groupe *A* vont exécuter la contrainte *A*, sans avoir l'obligation de se synchroniser entre eux. L'important est qu'une synchronisation ait lieu pour tous les clients appartenant à *A* et *B* lors du dernier point d'interaction. Là encore, en raison de possibilités d'exécutions divergentes du même contenu, il est impossible d'offrir une répartition hiérarchique.



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

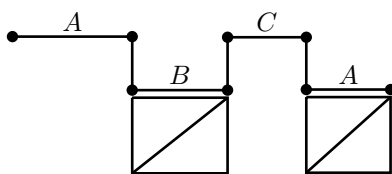
4.2.3. Cas non-interactif

On peut utiliser les méthodes de synchronisation décrites précédemment, mais une possibilité s'ajoute :

Comme on connaît les dates effectives auxquelles les objets sont sensés s'exécuter, on peut les fixer à l'avance sur chaque machine. Un parcours de graphe permet d'obtenir une estimation des dates minimales auxquelles il est possible de fixer des dates d'exécutions des éléments suivant un point d'interaction donné. Le même mécanisme a été utilisé dans [2] pour spécifier les dates d'exécution et de fin de flux audios.

- Avantage : s'il y a une déconnexion, l'exécution va continuer à fonctionner au moins jusqu'au prochain point d'interaction.
- Inconvénient : il suffit d'un peu de délai pour que le début de B survienne avant la fin de A. 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 toutes les autres machines. En autorisant cela, on perd donc une certaine tolérance au partitionnement.



Scénario 4 : Les groupes A, B, C exécutent des contraintes temporelles à la suite les uns des autres

4.2.4. Expressions et interactivité

Dans le cas où un scénario est exécuté intégralement en parallèle par différentes machines, il n'y a pas de problème : chacune vérifie les expressions en fonction des données dont elle dispose, et les valide à l'instant où elle le souhaite. C'est utile si par exemple on veut avoir plusieurs téléphones qui font tous tourner un scénario semblable, mais chaque individu peut choisir de faire avancer le scénario au rythme où il le souhaite.

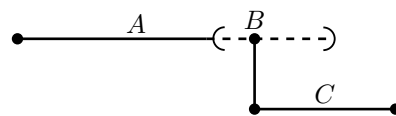
Dans le cas où on a un partage de certaines lignes temporelles par certaines machines, le problème de la synchronisation des expressions se pose.

- Prise de décision : 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 éléments. Plusieurs manières d'obtenir un consensus sont possibles :
 - Dans le cas des conditions, au moins une machine valide ou nie l'expression.
 - Dans le cas des points d'interaction, on peut fixer la valeur de l'expression à celle de la première machine qui la vérifie, à l'aide des estampilles.
 - Dans le cas des points d'interaction, une majorité de machines valident l'expression.
 - Toutes les machines valident ou nient l'expression.
- On peut supposer que des sous-parties d'un scénario pourraient être exécutées entièrement par une machine ou groupe.

On note plusieurs problèmes possibles :

- La déconnexion intempestive 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.

Ordonnancement : par estampilles.



Scénario 5 : Des groupes A et C exécutent les deux contraintes temporelles tandis qu'un groupe B atteint un consensus sur l'expression

Le scénario 5 peut se résoudre de plusieurs manières.

- Si l'exécution du nœud est résolue de manière asynchrone, toutes les machines de B envoient l'information de déclenchement à toutes les machines de A et C, sans garantie d'ordre. Les points d'interaction se déclenchent immédiatement. Note : si on veut garantir un ordre, ne peut-on pas faire : C stoppe A qui démarre B ? Ou bien si la borne max est atteinte, A stoppe C et déclenche B.
- Si l'exécution du nœud est résolue de manière

synchrone :

- Si la condition devient vraie, par consensus. On choisit une date, et on fixe l'arrêt des machines de A et le départ des machines de B par rapport à cette date, calculée en fonction des latences relatives. Potentiellement introduire un tick d'écart entre la fin de A et le démarrage de B ?
- Si on atteint la borne max : une machine est nécessairement la première à atteindre cette borne.

4.2.5. Consensus

Comme mentionné précédemment, nous nous trouvons face à plusieurs cas ou plusieurs clients doivent s'accorder sur le résultat d'une expression.

Paxos, Raft sont des algorithmes permettant de garantir un consensus.

S'il y a synchronisation, on sépare la synchronisation du consensus, de la synchronisation de l'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.

Note : abstraire la notion de consensus au sein d'un groupe.

Cas avancé et non traité ici : si un scénario est exécuté en parallèle par toutes les machines, chaque groupe peut avoir un consensus différent sur chaque point d'interaction. On peut rajouter un niveau d'indirection. Le plus général : chaque objet peut être exécuté par N groupes pouvant posséder chacun M clients. Si un client est dans deux groupes, comment gérer le conflit ? bof.

Cas synchrone : * Sous-expression devient vraie * Expression fixe une date possible pour le réseau et notifie les autres puis se fixe au bon délai.

Cas asynchrone : * Sous-expression devient vraie * Expression envoie un message aux autres et se lance immédiatement. Il faut définir sur quelle machine l'expression est validée.

- Cas de la borne max : point d'interaction, bof

Pour l'instant : group leader ? ou bien tous se communiquent l'information et prennent la décision en fonction de cette information ?

4.2.6. Cas particuliers

- Cas où groupe est vide : afficher un warning à l'auteur car ce n'est peut-être pas désirable et significatif d'un problème matériel.

- Cas de groupes avec un seul participant.

Si par exemple on dispose de deux clients exécutant des objets en série, la synchronisation peut être plus simple : on n'a pas besoin de l'étape de consensus.

4.3. Processus boucle

La notion de boucle a été introduite dans [1].

4.4. Récapitulatif

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

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.

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

- Les processus
- Les expressions
- Les contraintes temporelles

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

Pour le cas du processus scénario, la répartition peut avoir lieu de manière synchrone ou asynchrone, avec ou sans partage d'informations entre clients.

On remarquera que le problème de la répartition peut se voir comme deux cas :

- Exécutions en parallèle (les éléments d'un même groupe).
- Exécutions en série (les éléments d'un scénario qui se suivent).

5. SÉMANTIQUE

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

On fait le choix de décrire ce modèle via la sémantique existante de i-score ; en effet, on dispose déjà de primitives de synchronisation et d'envoi de message.

5.1. Introduction de primitives réparties dans i-score

TODO voir politiques d'exécution de réseaux de Petri dans thèse AA

5.1.1. Modélisation dans le cas non-interactif

Le premier cas étudié est celui du déplacement : deux contraintes C_1 , C_2 se suivent, on veut que la première s'exécute sur les clients d'un groupe A , et que la seconde s'exécute sur les clients d'un groupe B .



Scénario 6 : Deux contraintes temporelles se suivent

Dans le cas présenté par le scénario 6, 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 $C_{1 \rightarrow 2}$.
3. On déplace C_2 à la suite de cette contrainte temporelle.
4. Si on se trouve dans le cas asynchrone : on crée un message M_1 qui ira déclencher T_1 et on rend $C_{1 \rightarrow 2}$ souple. Dès qu'une machine de A arrivera à M_1 , le message sera envoyé.
5. Si on se trouve dans le cas précalculé : la durée calculée sera celle de $C_{1 \rightarrow 2}$. Les machines du groupe A et B peuvent être déconnectées après le début de la lecture et continueront à fonctionner comme prévu.

Quand il n'y a pas de point d'interaction, le cas synchrone n'offre pas d'avantages par rapport au cas précalculé.

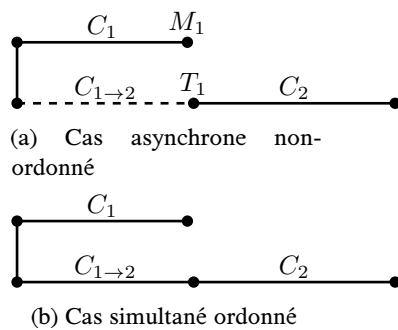


Figure 3 : Transformations pour l'exécution de deux contraintes en série sur deux machines séparées

- implémentation : on rajoute des délais dans les expressions.

- Introduire la possibilité pour une expression de faire une comparaison avec le wall clock.

Cas à traiter : - Trigger - Duplication - Déplacement avec plusieurs scénarios (consensus ?) -> introduire un device virtuel qui règle le consensus

5.1.2. Modélisation dans le cas interactif

On reprend ici le scénario 5, qui possède un point d'interaction, géré par le groupe B.

— T_B : consensus du groupe sur l'expression B

— $T_A : M_A, T_D : M_D, T_C : M_C$

Dans le cas asynchrone : M_A, M_D, M_C sont des messages qui vont déclencher les point d'interaction T_A, T_D, T_C dès qu'ils sont reçus.

Cas asynchrone ordonné : les messages partent le plus tôt possible, mais C, D , ne peuvent se déclencher avant la fin de A.

Cas synchrone (est-ce le bon mot ?) : on essaye de faire en sorte que les choses s'arrêtent et démarre le plus proche possible. Le scénario est le même que dans le cas asynchrone. La différence est dans les messages qui sont envoyés : M_A, M_D, M_C fixent la date à laquelle T_A, T_D, T_C doivent partir.

On notera qu'il est possible d'exprimer des variantes plus fines : par exemple synchroniser les fins mais pas les débuts. Néanmoins, ce travail a pour objectif de ne pas encombrer le compositeur d'une finesse trop peu souvent nécessaire, en lui offrant un accès facile à des concepts de haut-niveau dont la nécessité est apparue lors des études de cas.

5.2. Mécanismes de synchronisation

- Cas possibles :

* Synchronisation de la fin

* Synchronisation du début de la suite

6. ÉVALUATION

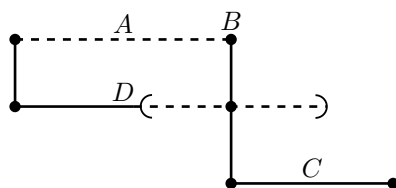
- Comparaison de l'algorithme "simple" et de l'algorithme avec retard

- Latence : prendre moyenne et écart-type sur les dix dernières valeurs ? Ou juste dernière valeur ? En LAN gigabit on est en général < à une milliseconde.

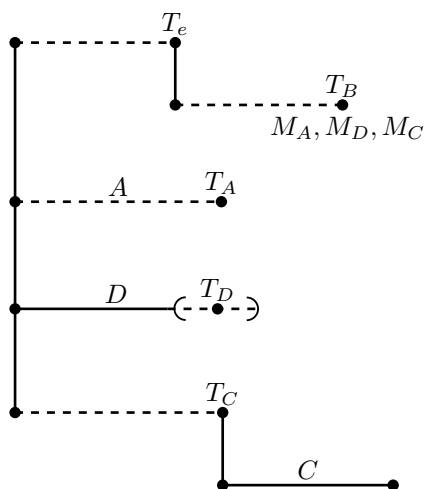
Références

- [1] Jean-Michaël Celerier, Myriam Desainte-Catherine et Jean-Michel Couturier. "Graphical Temporal Structured Programming for Interactive Music". In : *International Computer Music Conference*. 2016.
- [2] 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 Conference*. 2016.
- [3] Jean-Michaël Celerier et al. "OSSIA : Towards a unified interface for scoring time and interaction". In : *TENOR2015*. 2015.
- [4] Leslie Lamport. "Time, clocks, and the ordering of events in a distributed system". In : *Communications of the ACM* 21.7 (1978), p. 558–565.
- [5] Justin Sheehy. "There is no now". In : *Communications of the ACM* 58.5 (2015), p. 36–41.

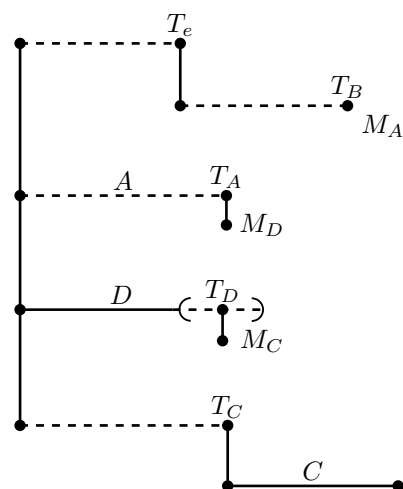
Annexe : politiques de répartition dans le cas interactif



(a) Un scénario possédant un point d'interaction que l'on veut répartir sur les clients des groupes A, B, C, D



(b) Répartition du scénario (a) dans le cas asynchrone : les contraintes associées aux groupes A, D et la contrainte associée au groupe C s'arrêteront et démarreront sans ordre prévisible.



(c) Répartition du scénario (a) dans le cas asynchrone ordonné : les contraintes associées aux groupes A, D et la contrainte associée au groupe C s'arrêteront dans l'ordre, avec un délai potentiel entre la fin de A, D et le début de C .