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 d'information entre musiciens en séparant une partition en portées ; la plupart des logiciels musicaux vont interpréter cette portée sur une seule machine, en gardant la possibilité de pistes avec des réglages indépendants.

Ce document présente d'une part une généralisation de cette notion de partage aux scénarios interactifs, et d'autre part une implémentation pratique dans le logiciel i-score, mettant en valeur les possibilités d'écriture augmentées 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 problématiques pratiques rencontrées 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

-> installation en son réparti ?

- Problème de la latence entre i-score et raspberry, + utilisation de bande passante.

- Cas des applis de téléphone : un objet qui s'exécute sur plusieurs machines en parallèle dont on veut aggréger les résultats

- Donner la possibilité de faire un dialogue (boucle avec un coup un point d'interaction qui vient de A, un coup un point d'interaction qui vient de B, etc)

- Approche alternative avec application mobile.

- Faciliter l'écriture de programmes qui s'exécutent sur des machines embarquées

- Lister les cas. D'abord non interactif (partage uniquement des processus), puis interactif.

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

2.1. État de l'art

2.1.1. Rappel du modèle d'i-score

Vocabulaire. Prendre article SMC.

Ici les processus de donnés sont représentés par une boite barrée en diagonale (cas courant d'une courbe montante).

Mettre accent sur hiérarchie

2.1.2. Répartition à l'exécution

- 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

2.1.3. Horloges

Problème de l'horloge :

— Si on utilise l'horloge système, pas de garantie qu'elle soit bien synchronisée. Et NTP pas dispo 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://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 méchanisme de vector clock approximatif).

Synchro d'horloges de deux machines dont une qui ralenti, si elles exécutent le même scénario ? Il faut les recaler 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 machine 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 <= 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.

2.1.4. Problématique de la sécurité

2.1.5. 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 example sur des cas simples.

La section 5 définit ces possibilités en utilisant les objets du modèle d'i-score.

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

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 (addresse 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 travaillent 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, de manière synchrone ou asynchrone.

L'intérêt d'un groupe est la tolérance aux pannes, déconnections, reconnections, 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.

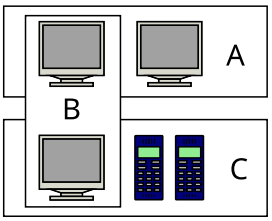


Figure 1: Plusieurs groupes avec plusieurs machines dans chaque groupe

3.2. Topologie

Maître-esclave pour l'édition.

Pour des raisons de simplicité, la première mise en œuvre se fait via une toplogie 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 ou des topologies différentes peuvent montrer un intérêt.

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

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

3.3. Fondements

Première approche basées sur réseau de Petri.

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

Il est possible d'implément ce méchanisme de répartition purement avec les primitives dont on dispose dans i-score, mais c'est prohibitoire pour les utilisateurs.

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

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

3.4. 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 : les scénarios i-score sont très compacts (en général quelques dizaines de kilo-octets).

3.5. Répartition de l'édition

- Partage de file undo - redo. Transformations opérationnelles.

- Autres approches : partage du modèle lui-même.

4. DESCRIPTION DE L'EXÉCUTION

On sépare les objets offrant une structure temporelle, des objets possédant des données (par exemple, une automation).

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 toutes les machines auquel il est assigné, et ne s'exécute pas pour les autres.

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.

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 évènement externe e se soit déclenché, visible par exemple en scénario 2) du cas plus simple ou les dates sont fixées, visible en scénario 3.

On travaillera dans les examples suivants avec trois groupes A, B, C disposant chacun d'un nombre inconnu de clients.

4.2.1. Modes de synchronisation

On identifie trois niveaux de synchronisation, qui peuvent chacuns avoir leur utilité dans un cas différent :

- 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.
- Synchrone : 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. Ce mode respecte la sémantique de i-score : les éléments s'exécutent dans le même ordre que si le scénario n'était pas réparti, au prix d'une latence augmentée. 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.
- Précalculé : ce mode est surtout utile 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. Plus la synchronisation des horloges sera fine entre les clients et plus ce mode 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.

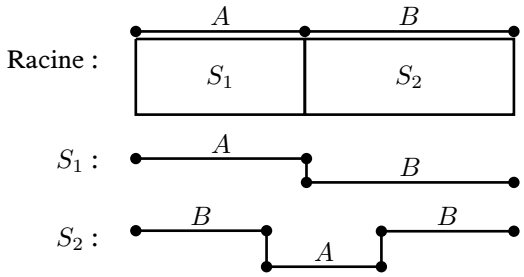
Lorsqu'un choix doit être fait, un consensus peut être pris au niveau du groupe auquel est assigné l'objet. Par exemple, quelle va être la vitesse à laquelle une contrainte temporelle va s'exécuter. Les mécanismes de consensus possibles sont détaillés plus en détail par la suite.

4.2.2. Cas interactif

Trois niveaux de partage :

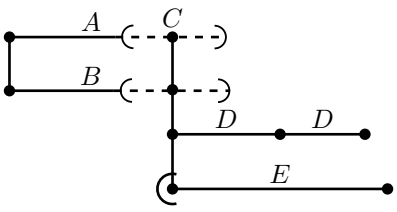
- Partage complet : il n'y a qu'une seule ligne temporelle partagée pour toutes les machines. Les annotations de machines indiquent l'emplacement d'exécution. Les groupes servent à indiquer ou non l'exécution d'un processus sur une machine et le groupe devant parvenir à un consensus pour une

expression donnée. Si par exemple la vitesse d'exécution d'une contrainte est modifiée en temps réel, cette modification est répercutée sur toutes les machines.
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 machines n'étant pas associées à ce processus ne l'exécutent pas, celles qui y sont associées l'exécutent toutes 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 machines 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 groupes assignés aux objets du scénario soient ignorés, récursivement : puisque chaque exécution va avoir des temps différents par conception, il ne peut pas y avoir de synchronisation.
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.

— Partage partiel : il peut y avoir plusieurs lignes temporelles dans un même scénario qui peuvent se resynchroniser à un instant donné. Les annotations donnent l'emplacement d'exécution, et de vérification des expressions. Une machine ne sait pas à l'avance si elle va pouvoir se déclencher ou non.

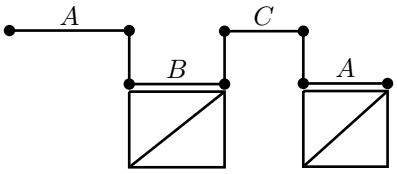
4.2.3. Cas non-interactif

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

Comme on connait 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 [*?*] 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 particulièrement important dans ce cas de garder les horloges 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 pert donc une certaine tolérance au partitionnement.



Scénario 3 : 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 ou 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 ou 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 ou il le souhaite.

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

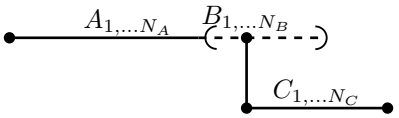
— Prise de décision : s'il n'y a qu'une seule ligne temporelle, chaque expression doit n'avoir qu'une valeur de vérité. Il doit donc y avoir un consensus

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

Le scénario 4 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

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 N clients. Si un client est dans deux groupes, comment gérer le conflit ? bof.

4.2.6. Simplifications

- Cas ou groupe vide - Cas ou groupe avec un seul participant

4.3. Processus boucle

La notion de boucle a été introduite dans *??*.

4.4. Récapitulatif

On choisit pour la répartition un modèle de document partagé : toutes les machines voient le même scénario.

On introduit une notion de client (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 groupe :
— 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

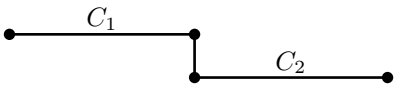
On décrit ici la sémantique des différentes méthodes de synchronisation vues précédemment.

On fait le choix de décrire ce modèle via la sémantique existante de i-score; cela peut être ramené à des réseaux de Petri assez facilement.

5.1. Introduction de primitives réparties dans i-score

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

5.1.1. Répartition des contenus dans le cas non-interactif



Scénario 5 : Deux contraintes temporelles se suivent

— Cas partagé : toutes les machines vont exécuter l'ensemble du scénario en parallèle.
— Cas synchronisé : chaque machine va exécuter une partie du scénario : par exemple, une machine exécute C_1 et une autre C_2.

- implémentation : on rajoute des délais dans les expressions.

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 ?

- Pour l'édition temps-réel, on doit permettre d'appliquer des "filtres" (par exemple qui vont rajouter une sur-expression, etc)

5.2. Méchanismes de synchronisation

- Cas possibles :

C1 ——t—— C3
C2 ———|——— C4

* Mode free ou synchronisé : pour chaque client ?

Cas 1 :

C1 ——t—— C3

C1 libre, C3 libre : dès que t est vrai, un message est envoyé au master qui envoie à C3 le point d'interaction. C1 s'est déjà arrêté. Voire, ils peuvent se trig eux même en asynchrone.

C1 synchro, C3 synchro : dès que t est vrai, C1 calcule la date minimale à laquelle C3 peut être notifié, envoie le message et fixe ce temps de son côté.

Cas 2 :

C1 ——t—— C3
C2 ———|

* Synchronisation de la fin
* Synchronisation du début de la suite

En pratique, on n'implémente que la possibilité de synchro / désynchro toute une time node; dire qu'une granularité plus fine est possible mais que l'intérêt n'apparaît pas en pratique dans les applications (et complexifierait l'UI pour rien).

Question principale : pour un time-node, comment choisit-on sur quelle machine une condition doit être vérifiée ? Possibilité de conditions groupées : faut-il que ce soit interne au formalisme (i.e. une case à cocher) ou externe (on dit explicitement machine1:/truc && machine2:/truc)

5.2.1. Cas ou il y a plusieurs machines dans un groupe

Attributs d'une expression :
— Qui vérifie cette expression : chaque machine individuellement, toutes les machines d'un groupe, n'importe quelle machine d'un groupe

Séparer les expressions et les time-node : notamment pour le cas de la borne max, ou on doit avertir d'autres machines par la suite.

Deux axes :

Décision \ Sync	Oui	Non
Locale	1	2
Partagée	3	4

— Décision locale : Pour un point d'interaction, soit tout le scénario est local (donc même contrainte avant / après et certains ordis n'exécutent pas du tout ce scénario), soit le premier à avoir un résultat avertit les suivants. -> Le résultat des autres machines n'influence pas l'exécution sur une machine donnée.
— Décision partagée : tous les éléments d'un groupe doivent prendre une décision avec une politique donnée : soit tous doivent vérifier la condition, soit un seul.
— Décision synchrone : On veut que les éléments s'arrêtent et démarrent le plus proche possible d'une même date en wall clock.
— Décision asynchrone : On veut que les éléments s'arrêtent et démarrent le plus vite possible dès qu'une information est disponible.

1. Décision locale synchrone : pas de sens
2. Décision locale asynchrone : pas de sens...

3. Décision partagée synchrone : toutes les machines attendent qu'un unique choix soit effectué pour le résultat de l'expression
4. Décision partagée asynchrone

Extension via système de composants

On ne synchronise qu'à chaque point d'entrée / de sortie d'un groupe de contraintes / time nodes

5.3. Extension des possibilités d'écriture

* Paramètres partagés
* Pattern matching sur addresses

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.

7. CITATIONS

Toutes les références bibliographiques des citations devront être listées dans la section "References", numérotées et en ordre alphabétique. Toutes les références listées devront être citées dans le texte. Quand vous vous référez au document dans le texte, précisez son numéro [*?*].