

Formalisation des relations temporelles entre une partition et une performance musicale dans un contexte d'accompagnement automatique

José Echeveste, Arshia Cont, Florent Jacquemard, Jean-Louis Giavitto

▶ **To cite this version:**

José Echeveste, Arshia Cont, Florent Jacquemard, Jean-Louis Giavitto. Formalisation des relations temporelles entre une partition et une performance musicale dans un contexte d'accompagnement automatique : Accompagnement musical automatique. Modélisation des systèmes réactifs (MSR 11), Nov 2011, Lille, France. pp.109-124. hal-01156639

HAL Id: hal-01156639
https://hal.science/hal-01156639

Submitted on 27 May 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Formalisation des relations temporelles dans un contexte d'accompagnement musical automatique

Stratégies de synchronisation et gestion des erreurs

José Echeveste* — **Arshia Cont*** — **Jean-Louis Giavitto*** — **Florent Jacquemard****

** Ircam – UMR 9912 CNRS & UPMC, Équipe Représentations Musicales*
1, place Igor-Stravinsky
75004 Paris

{Arshia.Cont, Jose.Echeveste, Jean-Louis.Giavitto}@ircam.fr

*** INRIA & LSV (UMR CNRS, ENS de Cachan)*
61, avenue du Président Wilson
94230 Cachan

florent.jacquemard@inria.fr

RÉSUMÉ. *Nous présentons les problématiques temps réel de l'accompagnement musical automatique vu comme un système réactif. Nous introduisons un modèle de datation prenant en compte les différentes échelles de temps spécifiques à la musique, et nous proposons plusieurs stratégies de synchronisation et de rattrapage d'erreurs effectivement utilisées dans diverses compositions musicales.*

ABSTRACT. *We sketch the real-time features required by automatic musical accompaniment seen as a reactive system. We formalize the datation of musical event taking into account the various temporal scales used in music. Various strategies for the handling of synchronization constraints and the handling of errors are presented.*

MOTS-CLÉS : *Informatique musicale, suivi de partition, accompagnement automatique, programmation synchrone, langage synchrone, rattrapage d'erreurs*

KEYWORDS: *Computer Music, Score Following, Automatic Musical Accompaniment, Synchronous Programming, Synchronous Language, Error Handling*

1. Introduction

Tout concert par définition est en temps réel. Le résultat d'une performance musicale correspond à la réalisation des événements décrits dans une partition de musique, exécutés en temps réel par des musiciens. L'interprétation des oeuvres musicales donne lieu à plusieurs réalisations possibles, souvent non déterministes, d'une même partition musicale. La figure 1 nous montre les fluctuations temporelles existantes entre l'interprétation possible d'un musicien et une réalisation « rigoureuse » de la partition.

L'interprétation musicale devient plus complexe quand plusieurs musiciens sont sur scène. Chaque musicien est responsable de l'interprétation de son propre texte (sa partition) en cohérence avec celles des autres. Il est donc question ici de synchronisation et coordination entre plusieurs agents, robustes aux variations possibles dues à l'interprétation. Dans une formation, les musiciens vont alors utiliser différents moyens pour partager un même « temps musical ». Chacun est capable d'anticiper le futur grâce à un tempo intérieur, qui évolue en temps réel en fonction des informations passées et présentes. C'est ensuite par le biais d'événements repères, qu'ils vont pouvoir se synchroniser entre eux. Cette synchronisation est assurée soit implicitement par les musiciens eux mêmes dans le cas d'un petit ensemble de musique (un quatuor à cordes par exemple), ou bien par un chef d'orchestre quand il s'agit d'un grand nombre de musiciens.

L'ordinateur peut remplacer un musicien dans un contexte de performance en temps réel, en apportant toutes les possibilités offertes par l'informatique musicale en matière notamment d'analyse/synthèse du son et d'aide à la composition. Il est alors nécessaire de synchroniser en temps réel une partie d'accompagnement (partition électronique) avec une ou plusieurs parties instrumentales, en écoutant le jeu du ou des musiciens.

Pour résoudre ce problème, la machine doit être capable d'exécuter deux tâches principales en parallèle (Cont, 2008) :

– écouter et reconnaître la partie jouée par l'instrumentiste ;

– synchroniser l'accompagnement en réaction à la partie jouée.

La première tâche est souvent nommée « *suivi de partition* » et pose des défis importants en traitement du signal et apprentissage automatique. Elle est chargée de convertir un flux audio en un flux symbolique (suite d'événements). Cette problématique est actuellement bien maîtrisée. La description de ce processus sort du cadre de cet article où elle sera vue comme une boîte noire. La deuxième tâche est proche de la problématique des systèmes réactifs temps réel, où plusieurs agents collaborent pour aboutir à un résultat synchrone, temps réel et déterministe. Bien que la première tâche ait été abondamment étudiée dans la littérature, la deuxième tâche reste un problème important et peu traitée. Nous proposons ici d'aborder cette problématique.

La section 2 présente le contexte du problème du suivi de partition à partir du système de reconnaissance d'Antescofo, cadre de notre travail. Antescofo est considéré

comme l'état de l'art en terme de performance et modélisation dans ce domaine (cf. (Cont, 2010) pour une description plus complète de ce système). Nous présentons les originalités de cette application dans le contexte des systèmes réactifs temps réels, et nous nous focaliserons sur la gestion du temps (temps relatif, temps hétérogène, tempo) dans la synchronisation de l'accompagnement. Les sections 3 et 4 décrivent le langage synchrone utilisé dans Antescofo et les problématiques de synchronisation et du traitement des erreurs rencontrées dans ce contexte. La section 5 définit un modèle de datation permettant de calculer les ordonnancements possibles des actions en fonction du jeu du musicien. Enfin nous aborderons dans la section 6 les perspectives futures en intégrant différents travaux.

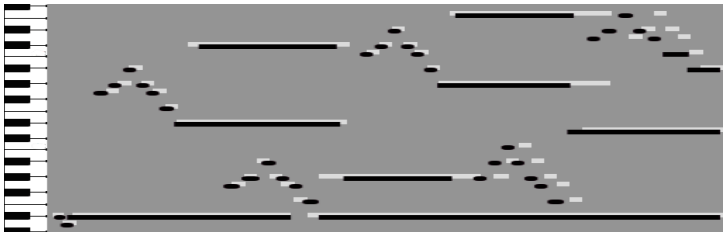


Figure 1. *Représentation graphique d'un extrait musical interprété par un musicien (en noir) par dessus la représentation du même extrait joué par un séquenceur MIDI (en gris clair)*

2. Contexte

2.1. *Antescofo, un suiveur de partition*

On définit traditionnellement le *suivi de partition* comme l'alignement automatique et en temps réel, d'un flux audio joués par un ou des musiciens, au sein d'une partition musicale symbolique. C'est avec l'avénement du standard MIDI que sont apparues les premiers suiveurs de partition dans les années 80 (Dannenberg, 1984), (Vercoe, 1984). Une décennie plus tard, les progrès en traitement du signal ont permis l'utilisation de systèmes d'analyse audio temps réel, lors de performances musicales. A la fin des années 90, ces techniques ont été combinées avec des méthodes probabilistes, améliorant ainsi les performances de ce type de système.

Antescofo étend le paradigme du stricte *suivi de partition*. Il offre au compositeur un langage expressif lui permettant de contrôler le lancement d'actions d'accompagnement (par exemple l'émission d'un son électronique), à partir des paramètres et des événements de la performance temps réel. La complexité croissante des processus électroniques mis en jeu, c'est-à-dire des programmes polyphoniques exécutés en concurrence et parallèlement au jeu du musicien, avec une gestion fine du temps musical, a naturellement conduit à la mise en place d'un langage synchrone permettant de spécifier ces interactions.

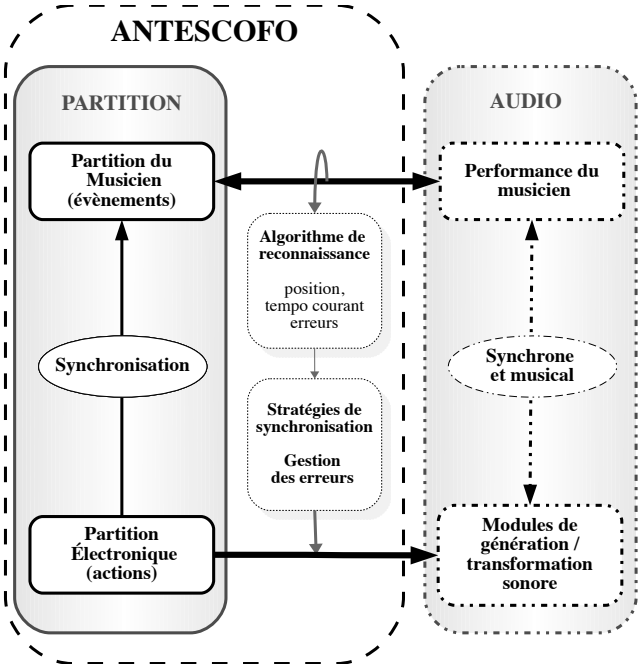


Figure 2. *Schéma global d'Antescofo au cours d'une performance*

2.2. *Antescofo, un système interactif, réactif et synchrone*

On peut considérer le système de suivi comme un système qui interagit avec le musicien et qui est toujours prêt à réagir à la vitesse imposée par ce dernier. Ces systèmes sont donc des systèmes *réactifs*, *synchrones*, *déterministes* et assurent une certaine sûreté de fonctionnement. Tous les événements et actions d'une partition partagent la même échelle de temps discrète et peuvent être datées sur cette échelle. De manière similaire à l'approche synchrone, on considère idéalement que les temps de calculs et de communication sont nuls : on peut appliquer le principe de de simultanéité (Halbwachs, 1998). Dans la réalité, le système doit réagir suffisamment vite aux entrées, pour prendre en compte tous les événements extérieurs dans le bon ordre et pour que la perception auditive de simultanéité soit conservée (de l'ordre de la dizaine de millisecondes).

2.3. *Les différents temps musicaux*

Pour que l'ordinateur puisse jouer avec d'autres musiciens, il est nécessaire de modéliser certaines des caractéristiques essentielles du temps musical. On appelle *tempo*

la vitesse, l'allure ou encore le mouvement d'exécution d'une œuvre musicale. Le tempo se distingue en musique du « temps » (unité de mesure). C'est ainsi qu'un tempo rapide détermine des temps courts tandis qu'un tempo lent détermine des temps longs.

Il existe dans Antescofo un modèle explicite du temps utilisé aussi bien au niveau de la reconnaissance des événements de la partition, qu'au niveau de la génération des actions d'accompagnement. Ce modèle repose sur un algorithme de reconnaissance de tempo (Large, 2001) et sur une catégorisation particulière des événements musicaux (Boulez, 1964).

De manière analogue à un musicien, qui se fie à un tempo intérieur pour anticiper le futur, la partie *réactive* du système a accès une valeur approchée du tempo courant au cours de la performance. L'algorithme de reconnaissance de tempo d'Antescofo est un algorithme anticipatif basé sur des études cognitives (le tempo est considéré comme une capacité d'attente dynamique évoluant de façon continue). L'étude de cet algorithme n'est pas l'objet de cet article, mais nous signalons aux lecteurs que l'estimation du tempo en temps réel est complexe et ne dépend pas seulement des deux derniers événements. Le suivi du tempo du musicien donne des informations complémentaires à la reconnaissance audio, pour la détection de la position dans la partition.

Un des atouts et une des originalités d'Antescofo, en tant que système réactif, réside dans la possibilité de dater des événements et des actions relativement au tempo, comme dans une partition classique, et donc de prendre en compte dynamiquement les fluctuations et les changements de tempo liés à l'interpretation. Le compositeur pourra choisir de synchroniser des actions sur des événements détectés ou bien après l'écoulement d'un certain délai, exprimé relativement au tempo.

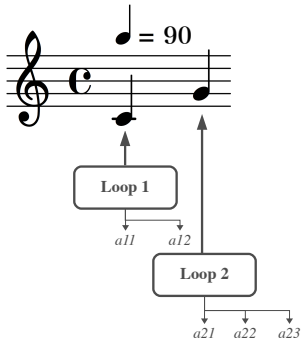
Nous insistons sur le fait que nous considérons et manipulons plusieurs échelles temporelles. Par exemple, la datation des événements musicaux lors de la *phase de composition* diffère de la datation des événements lors de la *phase d'interprétation* du musicien.

3. Langage et formalisation

Le langage d'Antescofo permet de spécifier conjointement la performance humaine et la description des événements électroniques. Nous souhaitons en effet que le compositeur puisse considérer les différentes parties électroniques de la même manière qu'une partie instrumentale classique. Les deux spécifications sont capables de décrire différentes échelles de temps hétérogènes (absolues, relatives, continues) au sein d'une même partition. La partition correspond à la description d'une performance « virtuelle » et « idéale ». Dans la section 5 nous distinguerons une telle performance d'une performance « réelle », soumise à l'élasticité du temps musical et aux erreurs éventuelles du système ou du musicien.

3.1. *Langage de la partie instrumentale*

La description de la *partition* jouée par le musicien précise les événements que le système sera amené à reconnaître (note, silence, accord...) ainsi que leur durée (souvent relativement au tempo). D'autres informations peuvent y apparaitre telles que les indications de changement de tempo : le compositeur spécifie un tempo idéal qui peut changer au cours de la partition. La figure 3 nous montre un exemple simple où l'on décrit deux événements successifs.




```
BPM 90
NOTE 60 1.0 note1
    Loop Loop1 4.0
    {  a11
       0.5 a12 }
NOTE 67 1.0 note2
    Loop Loop2 4.0
    { a21
      0.33 a22
      0.33 a23 }
```

Figure 3. *Extrait d'une partition et de sa description dans le langage d'Antescofo. La partition spécifie un tempo idéal de* 90 *pulsations par minute. Les actions (cf. section 3.2) sont indiquées sur fond grisé*

Nous utiliserons trois types de données pour spécifier cette partie instrumentale. $\mathcal{E}_S = \{e_1, \ldots, e_n\}$ désigne l'ensemble des n événements de la partition. $\mathcal{E}_S$ est muni d'une relation d'ordre total $<$ qui correspond au rang des événements dans la partition. $\mathcal{T}_S = \{0, t_1, \ldots, t_n\}$ dénote l'ensemble des *dates symboliques* associées à chaque événement dans la partition. Ces dates correspondent à une position dans la partition et qu'on exprimera relativement au tempo courant. La date 0 repère un événement *virtuel* : le début de la partition, indépendamment de tout événement. Par ailleurs, une interprétation associera à chaque t_i une valeur en millisecondes. L'ensemble $Tempo = \{T_1, \ldots, T_n\}$ représente l'ensemble des tempi idéaux associés à chaque événement. Par ailleurs, lors d'une interprétation, l'algorithme de reconnaissance associera à chaque événement un tempo estimé.

3.2. *Langage d'actions*

La deuxième partie du langage, la partition électronique, correspond au déclenchement d'actions à la suite d'événements ou d'autres actions. Ces actions sont ordonnancées en temps réel puis exécutées en temps voulu. Toute action est liée à un événement ou à une autre action et est lancée avec un délai (exprimée en temps relatif ou absolu) après la détection de l'événement, ou après le lancement de l'action, auquel elle est

liée. Nous ne décrivons ici que trois primitives, qui ne sont qu'une partie restreinte du langage, suffisante toutefois pour illustrer la problématique :

Action élémentaire. Une action élémentaire a correspond à une commande atomique envoyée à l'environnement (par exemple un module de synthèse sonore).

Groupe parallèle. La construction `Group` permet de regrouper logiquement au sein d'un même bloc plusieurs actions qui partagent des propriétés communes de synchronisation et de rattrapage des erreurs. Cette construction permet notamment la composition de phrases polyphoniques avec une gestion précise de la temporalité de ses éléments.

Groupe périodique La construction `Loop` est similaire à la précédente mais les actions qui la composent s'exécutent de façon périodique.

Une des spécificités du langage d'actions réside dans la propriété de composition hiérarchique ; les constructions (`Group` et `Loop`) peuvent s'imbriquer les unes dans les autres.

Dans l'exemple de la figure 3, il est indiqué que lorsque la `note1` est détectée alors deux `Loop` sont déclenchées. Toutes les deux ont une période de un temps. Le premier bloc est composé de deux actions élémentaires dont la deuxième est lancée avec un délai d'un demi temps après la première. Le deuxième bloc est composé de trois actions élémentaires avec des délais d'un tiers de temps entre chaque lancement.

Pour simplifier la présentation, on ne considérera que des délais $\mathcal{D}$ relatifs au tempo.

3.3. *Formalisation de la partition*

On définit la partition (*score*) correspondant à l'ensemble des événements et actions associées par la grammaire suivante :

$$\begin{array}{rcl} score & := & \varepsilon \mid event\ score \mid (d\ group)\ score \mid (d\ loop)\ score \\ event & := & e_i \quad \text{avec } e_i \in \mathcal{E}_S \\ group & := & \texttt{Group}\ synchro\ error\ (d\ action)^+ \\ loop & := & \texttt{Loop}\ synchro\ error\ p\ (d\ action)^+ \\ action & := & a \mid group \mid loop \end{array}$$

où $d \in \mathcal{D}$ représente un délai relatif au tempo (nombre de temps), ε est la séquence vide, $p \in \mathbb{R}$ dénote une période relative au tempo, $synchro$ est un attribut spécifiant la manière de gérer les synchronisation. et $error$ est un attribut spécifiant la manière de gérer les erreurs (cf. sect. 4).

L'événement ou l'action qui marque le début du délai avant l'exécution d'une action A est par défaut l'élément qui *précède syntaxiquement* A dans la partition.

4. Stratégie de synchronisation et de rattrapage d'erreurs

La performance du musicien est sujette à de multiples variations par rapport à la partition. Nous proposons différentes stratégies de synchronisation et de rattrapage d'erreurs au niveau de l'accompagnement, pour prendre en compte ces variations et gérer les éventuelles erreurs du musicien et de l'algorithme de reconnaissance.

4.1. *Stratégie de synchronisation*

Les constructions *group* et *loop* permettent de lancer une séquence d'actions à partir d'un événement déclenchant. Si les délais entre les actions, à l'intérieur d'un bloc, sont notés de façon relative, alors les dates de lancements des actions s'adaptent au tempo. Cependant la connaissance du tempo ne suffit pas pour une synchronisation précise avec les événements joués par le musicien (ce qui n'est pas toujours approprié, cf. section 5.2). Si le musicien ralentissait ou accélérait, des actions à l'intérieur du groupe serait temporellement décalé avec les événements ayant pourtant des instants relatifs égaux ; le tempo calculé n'est en effet qu'une prévision du futur (cf. figure 4).

Pour une synchronisation plus fine, on propose au compositeur d'assigner l'attribut `tight` aux blocs *group* et *loop* (on dira par la suite qu'un bloc est `loose` en l'absence de `tight`). Si le bloc est `tight`, toute action à l'intérieur sera déclenchée à partir de l'événement qui lui est immédiatement antérieur dans la partition (indépendemment de la structure de bloc). Ainsi le compositeur n'a pas besoin de segmenter son bloc d'actions pour spécifier les points de synchronisations ; cela lui permet de garder une vision de haut-niveau lors de la phase compositionnelle (en particulier pour les *loop*).

Dans notre modèle abstrait, tous les blocs d'action se verront associé un attribut *synchro* défini de la façon suivante : $synchro =$ `loose` $\mid$ `tight` .

4.2. *Gestion des erreurs*

Malgré les très bonnes performances de l'algorithme de reconnaissance, différents cas d'erreur peuvent être rencontrés : l'ordinateur peut confondre un événement avec un autre, ne pas détecter un événement, confondre un non-événement avec un événement. Il faut également considérer les erreurs possibles du musicien qui peut sauter un événement, jouer une fausse note ou jouer une note qui n'est pas dans la partition.

Dans tous les cas, on souhaite non seulement que le système continue à fonctionner, mais aussi qu'il réagisse le plus musicalement possible. L'algorithme de reconnaissance se charge de détecter les erreurs possibles du musicien. La partie concernant la synchronisation ne gère finalement qu'un seul type d'erreur : un événement n'a pas été détecté.

On offre la possibilité au compositeur de spécifier si une action est *globale* (attribut `global`) ou *locale* (attribut `local`) à un événement. L'idée est qu'une action

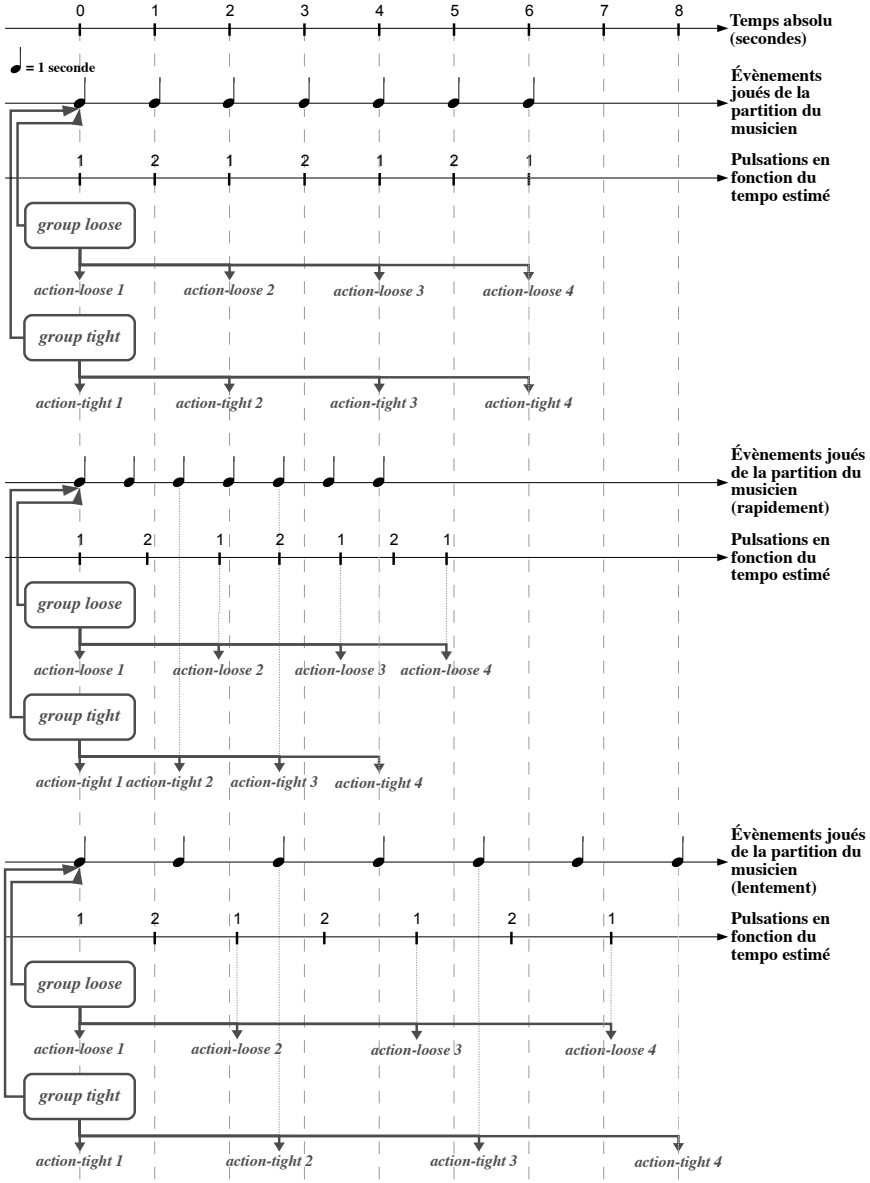


Figure 4. *Exemple d'une même partition jouée « idéalement », plus rapidement et plus lentement que le tempo spécifié*

`global` sera lancée (éventuellement en retard) même si l'événement associé n'est pas détecté, alors qu'une action `local` ne sera jamais lancée si l'événement auquel elle correspond n'est pas détectée.

Dans notre modèle, on associera à tous les blocs un attribut $error$ qui spécifie le caractère `local` ou `global` des actions qui le composent : $error = \texttt{global} \mid \texttt{local}$.

4.3. *Comportements en cas d'erreur*

La stratégie de rattrapage d'erreurs ($error \in \{\texttt{global}, \texttt{local}\}$) se combine avec la stratégie de synchronisation ($synchro \in \{\texttt{loose}, \texttt{tight}\}$) pour donner quatre comportements possibles en cas d'erreur (cf. figure 5).

– `local` - `loose` : si l'événement associé au début du bloc ($e1$ sur la figure 5) n'est pas détecté, alors aucune des actions du bloc ne sont lancées.

– `global` - `loose` : si l'événement associé au début du bloc ($e1$) n'est pas détecté, alors le bloc est lancé dès la détection d'un événement postérieur ($e2$). Toutes les actions à l'intérieur du bloc conservent les mêmes rapports temporels.

– `local` - `tight` : l'événement associé au début du bloc ($e1$) n'est pas détecté ; alors les actions du bloc sont ignorées si leur position spécifiée est antérieure à la position du premier événement détecté ($e2$) ; les actions du bloc sont lancées comme prévu initialement si leur position est après ce dernier événement ($e2$).

– `global` - `tight` : l'événement associé au début du bloc ($e1$) n'est pas détecté ; alors les actions du bloc sont lancées avec un délai nul si leur la position est antérieur à celle du premier événement détecté ($e2$) ; les actions du bloc, dont la position est après ce dernier événement ($e2$), sont lancées comme prévu initialement.

5. Datation symbolique des actions

Nous présentons dans cette section, une fonction ***datation*** nous permettant de calculer une date pour toutes les actions élémentaires. Cela permettra ensuite de simuler une performance et de comparer les différents ordonnancements possibles en fonction de la date des événements, du tempo et des erreurs.

Plus précisément, le résultat de la fonction ***datation*** est un ensemble composé de couples de la forme $\langle t_i, e_i \rangle$, et de quadruplets de la forme $\langle t_i, a, d, E \rangle$ où

– e_i est un événement, a une action élémentaire,

– t_i un marqueur temporel (symbolique) associé à l'événement e_i ou associé à l'événement déclencheur de l'action a,

– d est le délai qui doit s'écouler entre l'événement déclenchant et le début de l'action a,

– E est un ensemble de triplets $\langle e_i, error, d \rangle$ contenant les informations nécessaires à la gestion des erreurs.

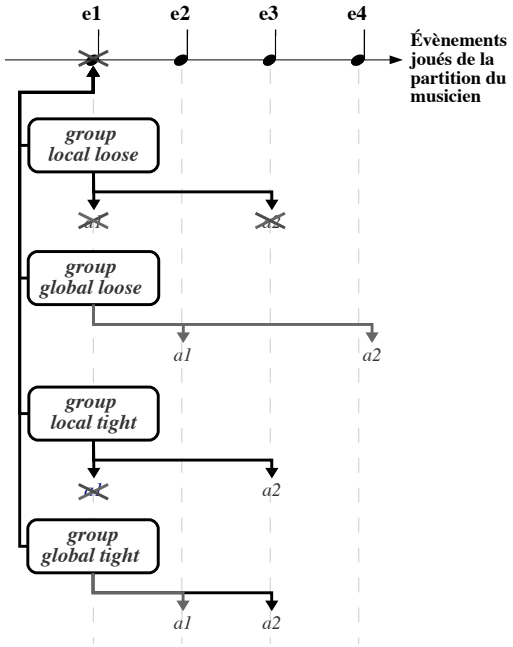


Figure 5. *Comportements possibles des blocs d'actions en cas d'erreur*

Un triplet $\langle e_i, \texttt{local}, d\rangle$ indique que l'action associée ne doit pas être exécuté si l'événement e_i n'est pas détecté. Le délai d est utilisé pour calculer la date à laquelle une action sera exécuté dans le cas d'un événement e_i manquant et d'une stratégie `global`.

5.1. *Définitions*

Dans les définitions qui suivent, les délais $d, d_i, \ldots$ correspondent aux actions $a, a_i, \ldots$. Si t_i est la date symbolique d'un événement e_i, on note t_{i+1} la date symbolique de l'événement e_{i+1} qui succède immédiatement à e_i dans la partition. On note $E + e$ pour $E \cup \{e\}$.

$$\begin{aligned}
&\textbf{\textit{datation}}[\![score]\!] = \textbf{\textit{datation}}_0[\![score]\!]\\
&\textbf{\textit{datation}}_t[\![\varepsilon]\!] = \varnothing\\
&\textbf{\textit{datation}}_t[\![e_i\ score]\!] = \{\langle t_i, e_i\rangle\} \cup \textbf{\textit{datation}}_{t_i}[\![score]\!]\\
&\textbf{\textit{datation}}_t[\![(d\ group)\ score]\!] = \textbf{\textit{date}}_{(t,\varnothing)}[\![d\ group]\!] \cup \textbf{\textit{datation}}_t[\![score]\!]\\
&\textbf{\textit{datation}}_t[\![(d\ loop)\ score]\!] = \textbf{\textit{date}}_{(t,\varnothing)}[\![d\ loop]\!] \cup \textbf{\textit{datation}}_t[\![score]\!]
\end{aligned}$$

La fonction $\textbf{\textit{date}}_{(\cdot,\cdot)}[\![\cdot]\!]$ est définie par les équations suivantes :

$$\textbf{\textit{date}}_{(t_i,E)}[\![d_a\ a]\!] = \big\{\langle t_i, a, d_a, E\rangle\big\}$$

$$\textbf{\textit{date}}_{(t_i,E)}[\![d_{\mathtt{G}}\ \mathtt{Group}\ \mathtt{loose}\ error_{\mathtt{G}}\ (d_1\ action_1)]\!] = \textbf{\textit{date}}_{(t_i,E+\langle e_i,error_{\mathtt{G}},d_1\rangle)}[\![d'\ action_1]\!]$$

$$\textbf{\textit{date}}_{(t_i,E)}[\![d_{\mathtt{G}}\ \mathtt{Group}\ \mathtt{loose}\ error_{\mathtt{G}}\ (d_1\ action_1)...(d_k\ action_k)]\!] = \textbf{\textit{date}}_{(t_i,E+\langle e_i,error_{\mathtt{G}},d_1\rangle)}[\![d'\ action_1]\!] \cup \textbf{\textit{date}}_{(t_i,E)}[\![0\ \mathtt{Group}\ \mathtt{loose}\ error_{\mathtt{G}}\ (d''\ action_2)\cdots(d_k\ action_k)]\!]$$

$$\textbf{\textit{date}}_{(t_i,E)}[\![d_{\mathtt{G}}\ \mathtt{Group}\ \mathtt{tight}\ error_{\mathtt{G}}]\!]\ (d_1\ action_1) = \begin{cases} \textbf{\textit{date}}_{(t_i,E+\langle e_i,error_{\mathtt{G}},0\rangle)}[\![d'\ action_1]\!] \\ \qquad \text{si } (t_i + d_{\mathtt{G}} + d_1) < t_{i+1} \\ \textbf{\textit{date}}_{((t_{i+1},E)}[\![0\ \mathtt{Group}\ \mathtt{tight}\ error_{\mathtt{G}}\ (d'''\ action_1)]\!] \quad \text{sinon} \end{cases}$$

$$\textbf{\textit{date}}_{(t_i,E)}[\![d_{\mathtt{G}}\ \mathtt{G}\ \mathtt{tight}\ error_{\mathtt{G}}\ (d_1\ action_1)\cdots(d_k\ action_k)]\!] = \textbf{\textit{date}}_{(t_i,E+\langle e_i,error_{\mathtt{G}},0\rangle)}[\![d'\ action_1]\!] \cup \begin{cases} \textbf{\textit{date}}_{((t_i,E)}[\![0\ \mathtt{Group}\ \mathtt{tight}\ error_{\mathtt{G}}\ (d''\ action_2)\cdots(d_k\ action_k)]\!] \\ \qquad \text{si } (t_i + d_{\mathtt{G}} + d_1) < t_{i+1} \\ \textbf{\textit{date}}_{((t_{i+1},E)}[\![0\ \mathtt{Group}\ \mathtt{tight}\ error_{\mathtt{G}}\ (d'''\ action_1)\cdots(d_k\ action_k)]\!] \\ \qquad \text{sinon} \end{cases}$$

avec $d' = d_{\mathtt{G}} + d_1$ et $d'' = d_{\mathtt{G}} + d_1 + d_2$ et $d''' = (t_{i+1} - t_i) - d_1 - d_{\mathtt{G}}$ et, en suivant notre convention, e_i l'événement correspondant au marqueur t_i.

$$\textbf{\textit{date}}_{(t_i,E)}[\![d_{\mathtt{L}}\ \mathtt{Loop}\ \mathtt{loose}\ error_{\mathtt{L}}\ p\ (d_1\ action_1)\ldots]\!] = \textbf{\textit{date}}_{(t_i,E)}[\![d_{\mathtt{L}}\ \mathtt{Group}\ \mathtt{loose}\ error_{\mathtt{L}}\ (d_1\ action_1)\ldots]\!] \cup \textbf{\textit{date}}_{(t_i,E)}[\![(d_{\mathtt{L}} + p)\ \mathtt{Loop}\ \mathtt{loose}\ error_{\mathtt{L}}\ p\ (d_1\ action_1)\ldots]\!]$$

$$\textbf{\textit{date}}_{(t_i,E)}[\![d_{\mathtt{L}}\ \mathtt{Loop}\ \mathtt{tight}\ error_{\mathtt{L}}\ p\ (d_1\ action_1)\ldots]\!] = \textbf{\textit{date}}_{(t_i,E)}[\![d_{\mathtt{L}}\ \mathtt{Group}\ \mathtt{tight}\ error_{\mathtt{L}}\ (d_1\ action_1)\ldots]\!] \cup \textbf{\textit{date}}_{(t_{sync},E)}[\![d_{sync}\ \mathtt{Loop}\ \mathtt{tight}\ error_{\mathtt{L}}\ p\ (d_1\ action_1)\ldots]\!]$$

avec $(t_{sync}, d_{sync}) = \mathit{find_sync}(t_i, d + p)$ et

$$\mathit{find_sync}(t,d) = \begin{cases} (t_i, d) & \text{si } (t_i + d < t_{i+1}) \\ \mathit{find_sync}(t_{i+1}, d - (t_{i+1} - t_i)) & \text{sinon} \end{cases}$$

Remarque : dans les équations définissant la datation des `Loop` nous n'avons pas explicité le cas d'arrêt qui correspond à $t_i = t_n$ pour ne pas alourdir la définition.

On obtient à partir des fonctions $\textbf{\textit{datation}}[\![\cdot]\!]$ et $\textbf{\textit{date}}_{(\cdot,\cdot)}[\![\cdot]\!]$, un ensemble de quadruplets de la forme $\langle t_i, a, d, Error\rangle$ et de couples de la forme $\langle t_i, e_i\rangle$. On peut maintenant ordonner temporellement cet ensemble dans le cas idéal et dans le cas d'une performance, en calculant les dates réels de chacun des événements et actions.

5.1.1. *Trace dans la cas idéal*

On calcule les dates de tous les événements et de toutes les actions élémentaires obtenues par rapport au tempo indiqué sur la partition. On rappelle que $Tempo = \{T_1...T_n\}$, est l'ensemble des tempi idéaux associés à chaque événement.

Pour un événement : $evt_evali(t_i) =$

$$\begin{cases} 0 & \text{si } t_i = t_0 \\ evt_evali(t_{i-1}) + (t_i - t_{i-1}) * T_{i-1} & \text{sinon} \end{cases}$$

Pour une action : $act_evali(\langle t_i, a, d, Error\rangle) =$

$$\begin{cases} evt_evali(t_i) + d * T_i & \text{si } (t_i + d) < t_{i+1} \\ act_evali(\langle t_{i+1}, a, (d - (t_{i+1} - t_i), Error\rangle) & \text{sinon} \end{cases}$$

5.1.2. *Trace dans le cas d'une performance*

Au cours d'une performance, les événements détectés $\mathcal{E}_R = \{e'_1, \ldots, e'_n\}$ sont associés aux dates réelles $\mathcal{T}_R = \{t'_1, \ldots, t'_n\}$. On note t'_i la date réelle de l'événement détecté e'_i. On a $\mathcal{E}_S = \mathcal{E}_R \cup \mathcal{E}_{Err}$, où $\mathcal{E}_{Err}$ est l'ensemble des événements qui n'ont pas été détectés et on a $\mathcal{E}_R \cap \mathcal{E}_{Err} = \varnothing$.

A chaque détection on estime le tempo courant. Soit $Tempo_R = \{T'_1 ... T'_n\}$, l'ensemble des tempi estimés associés à chaque événement e'_i détecté lors de la performance.

On note $\perp$ la date d'une action qui n'est pas jouée suite à une erreur d'interprétation ou de reconnaissance. Pour tout événement e_i de $\mathcal{E}_S$, on note $e_i \prec e'_j$ si e'_j est le premier événement détecté après l'événement e. Autrement dit, $e'_j \in \mathcal{E}_R$ et $t_i < t_j$ et pour tout e_k, $t_i < t_k < t_j \Rightarrow e_k \in \mathcal{E}_{Err}$. Si $e_i \prec e_j$, on note de même $t_i \prec t_j$.

Pour calculer la date de déclenchement d'une action, il faut calculer la durée réelle correspondant à un délai spécifié relativement au tempo. Le tempo réel varie au cours de l'interprétation et est réévalué par le système à chaque événement perçu. Il est donc nécessaire de prendre en compte tous les T'_j associés aux événements perçus e'_j par le système entre la date de déclenchement t'_i et l'échéance du délai (que l'on doit calculer !).

$act_evalr(\langle t, a, d_a, Error\rangle) =$

$$\begin{cases} \perp & \text{si } \exists\langle e, \texttt{local}, d\rangle \in Error \text{ et } e \in \mathcal{E}_{Err} \\ time_evalr(t_x, d) \text{ avec } t \prec t_x & \text{sinon si } \exists\langle e, \texttt{global}, d\rangle \in Error \\ & \qquad \text{et } e \in \mathcal{E}_{Err} \\ time_evalr(t_x, d_a) \text{ avec } t \prec t_x & \text{sinon} \end{cases}$$

avec $time_evalr(t_i, d) =$

$$\begin{cases} t'_i + d * T'_i & \text{si } t'_i + d * T'_i < t'_j \text{ avec } t'_i \prec t'_j \\ time_evalr(t_j, d - \frac{t'_j - t'_i}{T'_i}) & \text{sinon} \end{cases}$$

où $\frac{t'_j - t'_i}{T'_i}$ calcule la durée symbolique entre les deux événements e'_i et e'_j en fonction du tempo estimé T'_i. Notons que $time_evalr$ est appelé avec un temps symbolique t_i

correspondant à un événement qui est détecté, et donc pour lequel il existe une date réelle associée t'_i.

Nota Bene : si le calcul de la date réelle indiquée ci-dessus est correcte, la formulation employée n'est pas utilisable en pratique. En effet, le calcul de la date $t_i + d$ fait appel aux dates réelles t'_i et t'_j. Cette dernière n'est pas encore connue à la date t'_i lorsqu'on ré-estime la durée réelle du délai. L'implantation consiste à initier un chien de garde d'une durée de $d * T'_i$ à la date t'_i. Si un événement e'_j survient avant la levée du chien de garde, celui-ci est réinitialisé avec une durée mise à jour en fonction du tempo courant et du temps écoulé.

5.2. *Pertinence musicale des comportements de groupe*

Nous décrivons dans ce paragraphe des situations musicales simples, qui correspondent à chacun des comportements obtenus ci-dessus, pour mieux comprendre l'utilité de ces attributs au sein d'un bloc. On peut aussi caractériser l'imbrication des blocs en terme de comportement musical :

un bloc `local` **et** `loose` **:** Le bloc correspond à une phrase mélodique ayant une unité et une certaine indépendance rythmique ; si l'événement déclenchant le bloc n'est pas détecté, alors la phrase n'est pas jouée (ni en retard, ni morcelée).

un bloc `global` **et** `loose` **:** Le bloc correspond à un fond sonore indépendant s'étalant dans la durée. Il doit être déclenché même en retard ; un autre exemple pourrait être les initialisations des processus.

un bloc `local` **et** `tight` **:** Le bloc correspond à une harmonisation d'une mélodie ; si une note de la mélodie n'est pas détectée, seul l'accord associé à la note ne sera pas joué.

un bloc `global` **et** `tight` **:** Le bloc correspond à un enchaînement de notes tenues pendant un certain temps (accord arpégé), déclenchées de façon synchrone avec les événements. Si un événement n'est pas détecté, les notes seront tout de même déclenchées.

un bloc `loose` **à l'intérieur d'un bloc** `tight` **:** Le bloc `loose` garde une indépendance métrique une fois lancé par rapport à son groupe parent, dont tous les éléments sont synchrones.

un bloc `local` **à l'intérieur d'un bloc** `global` **:** Si l'événement qui déclenche le groupe parent n'est pas détecté, le bloc local n'est pas joué malgré son appartenance au groupe global.

6. Conclusion et perspectives

Nous avons présenté la problématique de l'accompagnement musical temps réel, détaillant les techniques de synchronisation et rattrapage d'erreurs qui sont implantées dans le système Antescofo (`http://repmus.ircam.fr/antescofo`). Une version

étendue de ce travail est présentée dans (Echeveste, 2011). Antescofo est un outil très apprécié des compositeurs, et réalisateurs en informatique musicale ; il sert d'ossature à l'organisation temporelle de nombreuses performances depuis 2007. Antescofo est utilisé régulièrement dans des oeuvres de compositeurs tels que Pierre Boulez, Philippe Manoury, Emmanuel Nuñes ou Marco Stroppa.

Des extensions, sollicitées par les utilisateurs, sont actuellement à l'étude, comme des groupes dynamiques (création et annulation), et la possibilité d'exprimer d'autres types de synchronisation, par exemple par contraintes, ce qui permettrait une écriture plus souple pour le compositeur. Par ailleurs, si dans cet article, nous avons adopté un point de vue volontairement dénotationnel, dans de futurs travaux, il pourrait être intéressant de proposer un modèle plus opérationnel basé sur un formalisme existant, par exemple des automates temporisés. Une perspective dans ce cadre serait la vérification formelle de propriétés qualitatives (faisant abstraction des marqueurs temporels), comme par exemple la préservation de l'ordonnancement de certaines actions quelque soit l'interprétation. Cette problématique est importante pour le compositeur et nous proposons pour finir quelques pistes pour l'aborder.

Les méthodes d'exploration exhaustive de modèles d'états finis (model checking) (Clarke *et al.*, 1999), requièrent en général des abstractions, particulièrement pour traiter des systèmes temps réel distribués tels que ceux utilisés par exemple dans les transports ou les systèmes industriels. Des techniques dédiées de model-checking temporisé permettent d'inférer des dépendances linéaires entre des paramètres temporels spécifiés dans des réseaux automates temporisés (Alur *et al.*, 1993). Ces paramètres correspondent typiquement, dans les cas cités, à des délais dus au temps de stabilisation de circuits, à la latence des canaux de communication, etc. Dans notre cas, ces paramètres temporels pourraient modéliser les délais liés à l'interprétation.

Des méthodes ont été mises au point (André *et al.*, 2009b) pour la synthèse de paramètres suivant une propriété à satisfaire. Le but d'une analyse est alors d'ajuster ces paramètres à de bonnes valeurs (André *et al.*, 2009a). Cette approche pourrait nous être utile, cependant, une modélisation de l'accompagnement nécessiterait un grand nombre de paramètres (a priori autant que de notes dans la partition), qui plus est non bornés, ce qui pourrait compliquer donc le passage à l'échelle. De plus, dans le cas de la vérification de circuits, l'imprécision (délais, décalages d'horloges) est souvent supposée bornée, ce qui réduit l'espace de recherche. Nous avons choisi ici, dans une première approche, de ne pas faire de telles hypothèses, laissant l'imprécision liée à l'interprétation musicale totalement indéterminée.

La notion d'un tempo fluctuant et indéterminé (évoluant en temps réel) pourrait aussi être capturée par un autre modèle de réseaux d'automates temporisés (Akshay *et al.*, 2008), avec des horloges évoluant indépendamment les unes des autres à des vitesses différentes non spécifiées. Dans ce cadre, on pourra s'intéresser à des propriétés de sureté ou de vivacité. Alternativement, nous pourrons aussi considérer de techniques plus efficaces d'analyse statique temporelle (timing-analysis) issues de problèmes comme le calcul du temps d'exécution maximal/minimal (WCET et BCET) (Wilhelm *et al.*, 2008), ou une sémantique du temps continu (Bertrane, 2006)

développée pour l'analyse par interpretation abstraite de systèmes distribués temps-réel avec des horloges désynchronisées.

Remerciements

Les auteurs remercient Gérard Berry, Laurent Fribourg et Marc Pouzet pour leurs suggestions, et leur soutien.

7. Bibliographie

Akshay S., Bollig B., Gastin P., Mukund M., Narayan Kumar K., « Distributed Timed Automata with Independently Evolving Clocks », *in Proc. of the 19th Int. Conf. on Concurrency Theory*, vol. 5201 of *Lecture Notes in Computer Science*, p. 82-97, Springer, 2008.

Alur R., Henzinger T. A., Vardi M. Y., « Parametric real-time reasoning », *Proc. of the twenty-fifth annual ACM symposium on Theory of computing*, STOC '93, p. 592-601, ACM, 1993.

André É., Chatain Th., De Smet O., Fribourg L., Ruel S., « Synthèse de contraintes temporisées pour une architecture d'automatisation en réseau », *in Actes du 7ème Colloque sur la Modélisation des Systèmes Réactifs (MSR)*, vol. 43 of *Journal Européen des Systèmes Automatisés*, Hermès, p. 1049-1064, 2009.

André É., Chatain Th., Encrenaz E., Fribourg L., « An Inverse Method for Parametric Timed Automata », *Int. Journal of Found. of Comp. Science*, vol. 20(5), p. 819-836, 2009.

Bertrane J., « Proving the properties of communicating imperfectly-clocked synchronous systems », *in* K. Yi (ed.), *Proc. of the 13th Int. Symposium on Static Analysis (SAS)*, vol. 4134 of *Lecture Notes in Computer Science*, p. 370-386, Springer, 2006.

Boulez P., *Penser la Musique Aujourd'hui*, Gallimard, 1964.

Clarke E. M., Grumberg O., Peled D. A., *Model Checking*, MIT Press, 1999.

Cont A., « ANTESCOFO : Anticipatory Synchronization and Control of Interactive Parameters in Computer Music », *Proc. of Int. Comp. Music Conference (ICMC)*, Belfast, August, 2008.

Cont A., « A coupled duration-focused architecture for realtime music to score alignment », *IEEE TPAMI.*, vol. 32, p. 974-987, June, 2010.

Dannenberg R. B., « An On-Line Algorithm for Real-Time Accompaniment », *Proc. of the Int. Comp. Music Conf. (ICMC)*, p. 193-198, 1984.

Echeveste J., *« Stratégies de synchronisation et gestion des variables pour l'accompagnement musical automatique »*, Master ATIAM, UPMC, Paris, 2011. Rapport disponible sur `articles.ircam.fr`

Halbwachs N., « Synchronous Programming of Reactive Systems. », *CAV'98*, p. 1-16, 1998.

Large E. W., « Periodicity, pattern formation, and metric structure », *Journal of New Music Research*, vol. 22, p. 173-185, 2001.

Vercoe B., « The Synthetic Performer in the Context of Live Performance », *Proceedings of the ICMC*, p. 199-200, 1984.

Wilhelm R. et al. , « The worst-case execution-time problem–overview of methods and survey of tools », *ACM Trans. Embed. Comput. Syst.*, vol. 7, p. 36 :1-36 :53, May, 2008.