MASTER ATIAM
Rapport de stage

José Echeveste

Stratégies de synchronisation et gestion des variables pour l'accompagnement musical automatique

Responsables *:*
Arshia Cont (Ircam) et Florent Jacquemard (INRIA Saclay)

Lieu *:*
Ircam - Equipe représentation Musical
1, place Igor-Stravinsky
75004 Paris

Dates du stage :
du 1er mars au 31 juillet 2011

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

Remerciements

Je voudrais tout d'abord exprimer ma sincère reconnaissance à mes encadrants Arshia Cont et Florent Jacquemard, mais aussi à Jean-Louis Giavitto et Carlos Agon pour la confiance qu'ils m'ont accordée, leurs conseils et leur disponibilité. Ce stage s'est déroulé de la meilleure façon qui soit. Je tiens à remercier Gérard Assayag et toute l'équipe Représentations musicales pour l'accueil qui m'a été réservé. Merci aux professeurs, coordinateurs et étudiants du Master ATIAM pour cette année riche et intense. Enfin je remercie l'Ircam et tout son personnel.

Résumé

Antescofo, système de suivi de partition, offre un langage synchrone pour la composition et l'écriture de l'interaction musicale. Après un bref exposé du contexte et des enjeux impliqués, le rapport présente les différentes stratégies de synchronisation et de rattrapage d'erreurs qui enrichissent le langage. Un modèle de datation prenant en compte plusieurs échelles de temps spécifiques à la musique, considère ces stratégies pour déduire les ordonnancements possibles des événements captés et actions générées. On y décrit également un prototype pour la gestion des variables qui augmente considérablement le pouvoir expressif du compositeur. Toute les nouvelles implantations ont été testées à partir d'exemple musicaux paradigmatiques, comprenant entre autres des compositions musicales utilisant Antescofo. Une évaluation de ces tests précède la conclusion et les perspectives.

Table des matières

Chapitre 1

Introduction

1.1. Cadre et déroulement du stage

Ce stage de fin d'étude s'est déroulé à l'Ircam au sein de l'équipe Représentations musicales dirigée par Gérard Assayag. L'Ircam (Institut de recherche et de coordination acoustique/musique) est une institution où se mêlent recherches scientifiques et musicales dans un contexte de création musicale contemporaine. L'équipe Représentations musicales mène des recherches et développements sur la formalisation informatique des structures musicales et sur les langages et paradigmes informatiques adaptés à la musique. Ces travaux aboutissent à des applications allant de la composition assistée par ordinateur (CAO) à la musicologie computationnelle en passant par les systèmes de suivi de partition.

Mon travail a été encadré par Arshia Cont (Ircam), Jean-Louis Giavitto (Ircam) et Florent Jacquemard (INRIA Saclay). Toutes les semaines, une réunion était organisée pour aborder de nouvelles problématiques, présenter les derniers résultats ou échanger des idées avec une personne extérieure à l'équipe lors de séminaires informels.

Après une première période de documentation et de réflexion, une phase de développement a rapidement donné des résultats interessants. Cela étant, un travail de formalisation a été entrepris afin de préciser nos idées, de raisonner sur le code, et d'ouvrir la voie à de nouveaux outils. Le résultat de cette réflexion a donné lieu à la rédaction d'un article qui a été accepté pour le colloque intitulé Modélisation des Systèmes Réactifs (MSR). Ce colloque se situe dans le cadre général de la modélisation, l'analyse et la commande des systèmes réactifs et temps réel. L'originalité et la spécificité du problème présenté dans l'article ont été particulièrement appréciés par les évaluateurs.

Une nouvelle période de documentation et développement suivirent avant de conclurent le travail de ce stage par de nombreux tests et validations.

1.2. Contexte

Tout concert par définition est en temps réel. Le résultat d'une performance musicale correspond à la réalisation des événements spécifiés 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 déterministe, d'une même partition musicale. La figure 1.1 nous montre les fluctuations temporelles existantes entre l'interprétation possible d'un musicien et une réalisation « rigoureuse » de la partition.

Figure 1.1 – 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)

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. 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. La deuxième tâche est proche de la problématique des systèmes réactifs temps réel, où plusieurs agents collaborent pour aboutir à un résultat synchrone[1], temps réel[2] et déterministe[3]. Bien que la première tâche ait été abondamment étudiée dans la littérature, la deuxième tâche reste un problème important et peu traitée.

Le sujet de ce stage se concentre sur cette seconde problématique 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 (Cont, 2010).

1.3. Problématiques et enjeux généraux du suivi de partition

On définit traditionnellement le *suivi de partition* comme l'alignement automatique et en temps réel, d'un flux audio joué par un ou des musiciens, au sein d'une partition musicale symbolique (cf. figure 1.2). 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.

On peut distinguer deux principaux champs d'applications pour l'utilisation du suivi de partition :

– l'accompagnement automatique (application pédagogique) ;

– la musique mixte (pour la création musicale contemporaine).

1.3.1. *Accompagnement automatique*

L'objectif des systèmes d'accompagnement automatique est d'abord d'écouter l'interprète en temps réel pour extraire les paramètres de position et de tempo par rapport à

1. fonctionnant au rythme d'une horloge commune

2. soumis a des contraintes de temps liés à son environnement physique

3. donnant un même résultat à même conditions d'entrée et code d'exécution

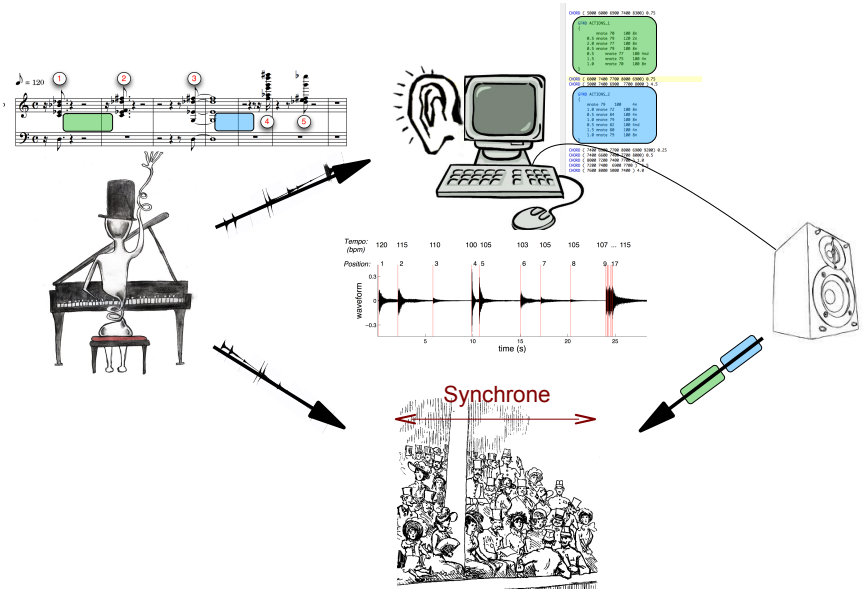


Figure 1.2 – Principe d'utilisation d'un système de suivi de partition

sa partition. La deuxième tâche a pour but d'exécuter en temps réel l'accompagnement et ce de manière synchrone au jeu du musicien. La partie d'accompagnement peut être décrite de façon symbolique puis synthétisée en temps réel (Vercoe, 1984; Dannenberg, 1984) ou bien elle peut être générée en utilisant un enregistrement audio traité en temps réel à l'aide de techniques de vocodeur de phase. Cette dernière technique est par exemple utilisée dans le logiciel Music Plus One, développé par l'équipe de Christopher Raphael (Raphael, 2011) de l'Université d'Indiana.

1.3.2. *Musique mixte*

Le deuxième type d'utilisation concerne ce que l'on appelle la musique mixte. Les premières oeuvres alliant instrumentation classique et électronique remontent aux années 50 avec des compositeurs tels que Bruno Maderna, Karlheinz Stockhausen et Mario Davidovsky ; ils utilisent alors la bande magnétique comme support audio pour la partie électronique. La bande magnétique fut en son temps une technologie interressante en ce qui concerne la synthèse des sons. Cependant sa rigidité et son déterminisme au niveau de l'organisation temporelle limitaient grandement les interactions possibles entre le musicien et l'électronique, le premier étant soumis au second.

Dans les années 80, certains compositeurs et scientifiques reconnaissent dans les premiers synthétiseurs en temps réel, un moyen d'assouplir ces contraintes temporelles. Combinés avec le suivi de partition, le musicien peut alors contrôler en toute liberté le temps musical, les processus électroniques étant générés par rapport à son jeu. La musique électronique peut désormais être interprétée.

Robert Rowe (Rowe, 1992; Rowe, 2004) montre comment ces nouveaux paradigmes ont affectés les différentes pratiques en informatique musicale. Philippe Manoury a été l'un des premiers compositeurs à intégrer ces techniques dans ses oeuvres à la fois comme un processus de composition que comme un outil de performance. Il collabore notamment avec Miller Puckette pour les compositions de Jupiter (1987) et Pluton (1988 – 1989). Ils développent ensemble de nombreux principes d'interactivité en temps réel, prémisses du l'environnement de programmation Max (Puckette, 1988). Manoury développe notamment le concept de partition virtuelle (Manoury, 1990).

1.3.3. *Partition virtuelle*

Une partition virtuelle est une organisation musicale dans laquelle on connait la nature des paramètres que l'on souhaite traiter, mais pas leurs valeurs exactes, ces dernières étant exprimées en fonction de la performance du musicien. Une partition virtuelle se compose donc de programmes électroniques fixés ou dépendants des paramètres de l'environnement extérieur. Autrement dit, un processus électronique existe dans la partition musicale, au côté de la transcription instrumentale, et son résultat est évalué au cours de la représentation en fonction de l'interprétation instrumentale. Un objectif de la partition virtuelle est d'intégrer à la fois les aspects liés à la performance et ceux liés à la composition assistée par ordinateur dans un même cadre d'écriture.

Le système de suivi de partition est donc responsable d'une communication entre l'ordinateur et le musicien, fidèle à la partition, permettant des interactions musicales complexes semblables à celles pouvant exister entre des musiciens.

Actuellement, le concept de la partition virtuelle est au cœur de la plupart des environnements de programmation interactif en musique assistée par ordinateur. Des pratiques non traditionnelles de l'informatique musical ont pu être intégrées, comme la composition interactive (Chadabe, 1984), la composition pour instrument augmenté (Machover *et al.*, 1989), l'improvisation composée (Chabot *et al.*, 1986). Manoury utilisera ces différentes pratiques, entre autres choses, dans plusieurs de ses pièces en temps réel (Puckette *et al.*, 1992).

Il est intéressant de noter que l'école temps réel a été l'objet de critiques constructives et intéressantes à ses débuts. Risset (Risset, 1999) et Stroppa (Stroppa, 1999) notamment, ont souligné un défaut au niveau de la considération compositionnelle et temporelle dans les environnements existants à l'époque. Ces critiques ont fortement influencé l'élaboration conceptuelle d'Antescofo.

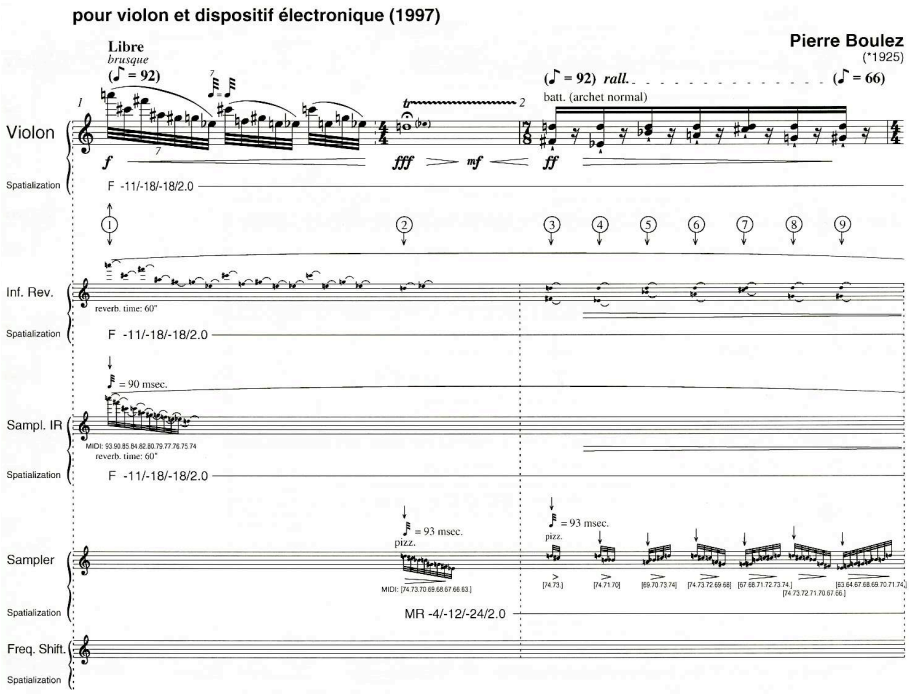


Figure 1.3 – Deux premières mesures de Antemes II composé par Pierre Boulez, pour violon et musique électronique (1997)

Nous tenons à souligner que la partition virtuelle étend la notion d'accompagnement musical automatique en ne se limitant pas aux notes et accords. Ces derniers pouvant être remplacés par des processus électroniques temps réel.

1.3.4. *Le suivi de partition en pratique*

Nous souhaitons illustrer ici nos propos en montrant l'architecture standard d'une pièce de musique mixte utilisant les techniques de suivi de partition. La figure 1.3 est un extrait de l'oeuvre de Pierre Boulez intitulée Anthèmes II pour violon et musique électronique composée en 1997. La partition correspond à la partie jouée par le violon et à une notation approximative de la partie électronique. Chaque système de cette dernière partie correspond à des processus électroniques spécifiques, pré-enregistrés ou calculés en temps réel, eux même contrôlés par des paramètres de spatialisation. Le séquencement de ces différents processus est aussi bien noté relativement au tempo de la performance que de façon absolu. Les nombres encerclés correspondent aux points de synchronisation entre la partie instrumentale et la partition électronique.

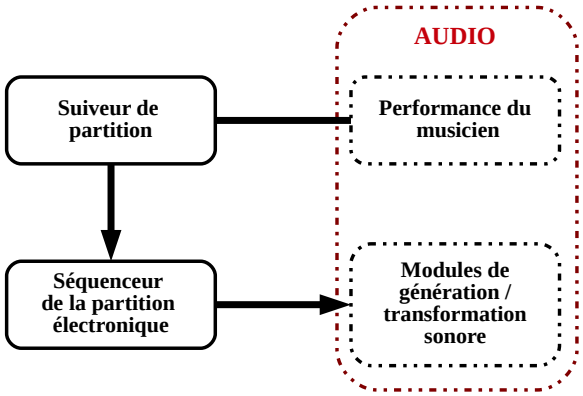


Figure 1.4 – Organisation classique d'une pièce de musique mixte.

La figure 1.4 résume l'architecture générale de la plupart des pièces de musique mixtes utilisant un suiveur de partition ou des environnements de programmation tels que Max/MSP (`http://www.cycling74.com`) et PureData (`http://puredata.info/`). Il montre bien la tendance générale qui consiste à séparer le couple musicien / suivi de partition de l'environnement permettant le séquencement des processus électroniques. En effet, la partie instrumentale et les points de synchronisations sont généralement spécifiés au niveau du suivi de partition qui se charge de l'alignement automatique. La partition électronique est quant à elle enregistrée sous forme de séquences de structures de données indexées (communément appelé qlists en Max/MSP et PureData). Elles sont rattachées à des index temporels symboliques qui correspondent à des points de synchronisation sur la partition électronique.

La modularité des environnements tels que Max ou PureData permettent la coexistence de processus sonores multiples dans un seul patch qui peut être contrôlé par la partition séquentielle électronique.

On peut distinguer deux objectifs distincts recherchés dans le développement du suiveur de partition. Le premier est scientifique : le but est d'éviter le maximum d'erreur au niveau de l'algorithme de reconnaissance. Le deuxième objectif est musical : faire en sorte que le résultat reste musicalement acceptable malgré les erreurs de reconnaissances et les erreurs du musicien.

1.4. Présentation du système Antescofo

Antescofo (cf. figure 1.6) étend le paradigme du stricte *suivi de partition*. Il offre au compositeur un langage expressif lui permettant de contrôler le lancement d'actions

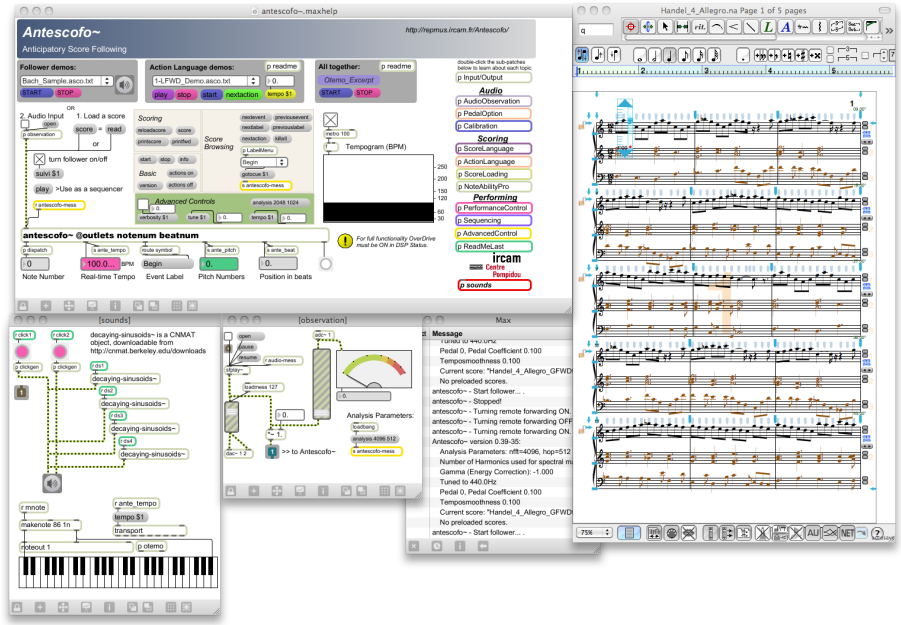


Figure 1.5 – Capture d'écran d'une session de travail avec Antescofo

d'accompagnement (par exemple l'émission d'un son électronique ou la commande d'un filtre), *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.

1.4.1. *Antescofo, un système réactif et synchrone*

On peut considérer la partition électronique et son programme correspondant comme un système informatique réagissant aux données issues de l'analyse du système de reconnaissance. Sa vitesse d'exécution dépend directement des données de son environnement. Antescofo peut donc être assimilé aux systèmes *réactifs*, *synchrones*, *déterministes* et qui comme eux, est supposé assurer une certaine sûreté de fonctionnement. Tous les événements et actions d'une partition partagent la même échelle de temps discrète et peuvent être datés 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 simultanéité (Halbwachs, 1998).

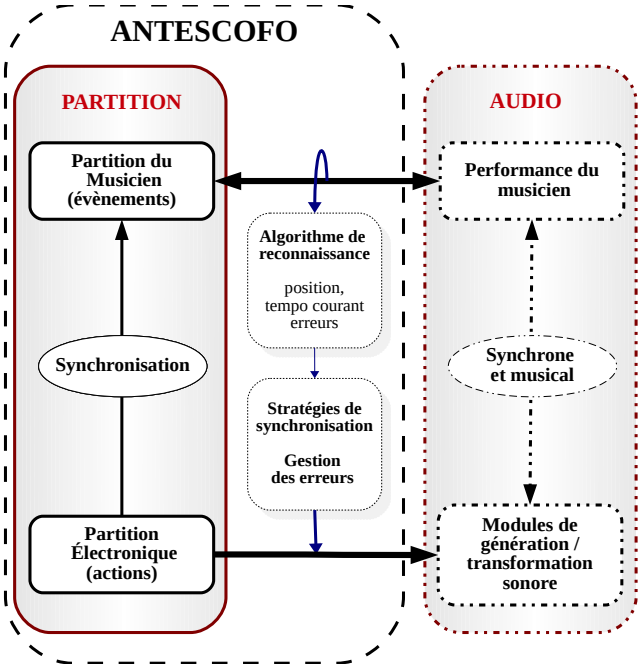


Figure 1.6 – Schéma global d'Antescofo au cours d'une performance

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).

Le tableau 1.1 nous montre l'analogie que l'on peut établir entre composition pour musique mixte avec suivi de partition et programmation synchrone. Dans une partition, un compositeur doit en effet prévoir et spécifier l'évolution temporelle d'une série de processus et les événements de la performance musicale (les entrées) dont ils dépendent. De la même façon qu'un ingénieur utiliserait un langage synchrone pour spécifier les commandes à envoyer à des actionneurs, en fonctions des données acquises grâce à des capteurs, le compositeur décrit les réactions musicales du système par rapport à son environnement (les événements reconnus).

1.4.2. *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

Musique mixte	**Paradigme synchrone**
partition	programme
composer	programmer
performance musicale	exécution
partition instrumentale	entrée attendue
partition interprétée (flux audio)	entrée réelle
actions électroniques	sortie attendue
actions jouées	sortie réelle

Tableau 1.1 – Analogie entre l'écriture de l'interaction musicale et le paradigme synchrone.

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; Xenakis, 1967).

De manière analogue à un musicien, qui se fie à un tempo intérieur pour anticiper le futur, la partie *réactive* du système a accès à 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). Nous signalons aux lecteurs sans entrer dans les détails 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 suiveur de partition et 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'interprétation. 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.

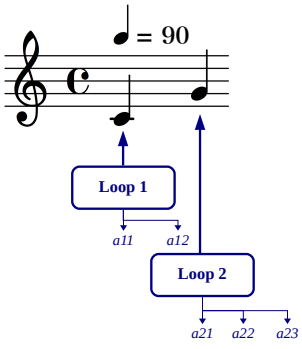
Nous insistons sur le fait que nous considérons et manipulons plusieurs échelles temporelles. Par exemple, la datation des événements musicaux lors de la *phase de composition* diffère de la datation des événements lors de la *phase d'interprétation* du musicien.

1.4.3. *Un langage pour la composition*

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.

1.4.3.1. *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 1.7 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 1.7 – 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 1.4.3.2) sont indiquées sur fond grisé.

1.4.3.2. *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 à 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 arbitrairement, les unes dans les autres.

Dans l'exemple de la figure 1.7, il est indiqué que lorsque la `note1` est détectée alors deux `Loop` sont déclenchées. Toutes les deux ont une période de 4 temps. Le premier bloc est composé de deux actions élémentaires, 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.

1.5. Motivations du stage

L'objectif de ce travail consiste à enrichir le langage afin d'offrir de nouvelles stratégies de synchronisation et rattrapage d'erreurs entre les événements et les actions. Cela donne la possibilité aux compositeurs de spécifier plus finement le déroulement du temps musical au sein des bloc d'actions, tout en décrivant leur comportement en cas d'erreur du système ou du musicien. De plus, on introduit une formalisation de la partition couplée avec une série de fonctions de datation permettant d'étudier les différents ordonnancements possibles en fonction de l'interprétation du musicien. Il est également question de la mise en place d'un prototype pour la gestion des variables. Les résultats implantés dans le système Antescofo sont validés sur des exemples musicaux réels.

Chapitre 2

Étude, formalisation et développement d'un langage pour l'écriture de l'interaction musicale

La partition correspond à la description d'une performance « virtuelle » et « idéale », c'est à dire sans aucune erreur et avec un tempo égal au tempo spécifié. Nous distinguerons une telle performance d'une performance « réelle », soumise à l'élasticité du temps musical (introduite par l'interprétation) et aux erreurs éventuelles du système ou du musicien. Nous présenterons dans ce chapitre, les différentes stratégies de synchronisation et de rattrapage d'erreurs qui définissent des rapports temporels spécifiques entre les événements captés et les actions générées ; une fonction de datation nous permettant de calculer une date pour toutes les actions élémentaires. Cela permet de simuler des performances « réelles » et de comparer les ordonnancements possibles des événements et actions.

2.1. Formalisation de la partition

Nous utiliserons trois types de données pour spécifier la 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. Un événement est l'occurrence d'une note, un accord, un silence... et sera produit par un musicien lors de l'interprétation.

– $\mathcal{T}_S = \{0, t_1, \ldots, t_n\}$ 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 et spécifiés par le compositeur. Par ailleurs, lors d'une interprétation, l'algorithme de reconnaissance associera à chaque événement un tempo estimé.

Pour simplifier la présentation, on ne considèrera que des délais $\mathcal{D}$ relatifs au tempo (dans Antescofo les durées peuvent aussi être exprimés en temps absolu).

On définit la syntaxe de 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. 2.2).

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. Cf. par exemple la figure 1.7 : l'action qui précède a22 est l'action a21 et l'événement qui précède la loop Loop2 est la NOTE 67.

2.2. 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.

2.2.1. *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 détecté. 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 2.2.4). 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 2.1).

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 syntaxique 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*).

Plusieurs approches ont été envisagées pour résoudre le problème de la recherche de points de synchronisation. Une première approche statique consistait à segmenter les *group* (bloc non-périodique) munis de l'attribut `tight` pendant l'analyse syntaxique. On recherchait alors pour chaque message les points d'accroches les plus appropriés. Une deuxième approche dynamique a été adopté pour les *loop* (bloc périodique). Elle consistait à rechercher, à chaque début de période, un point de synchronisation pour le déclenchement de la prochaine itération. On pourrait assimiler cela à une sorte d'évaluation paresseuse. L'approche dynamique a ensuite été généralisée à tous les blocs où l'on recherche pour chaque élément du bloc le point de synchronisation le plus approprié. Cette recherche pendant l'exécution du programme simplifie la gestion des hiérarchies des groupes ainsi que la gestion des erreurs. De plus, elle devient nécessaire lorsque les délais et les périodes sont spécifiés à l'aide de variables pouvant changer de valeur dynamiquement (cf. 2.4).

Dans notre modèle abstrait, tous les blocs d'action se verront associé un attribut *synchro* défini de la façon suivante :

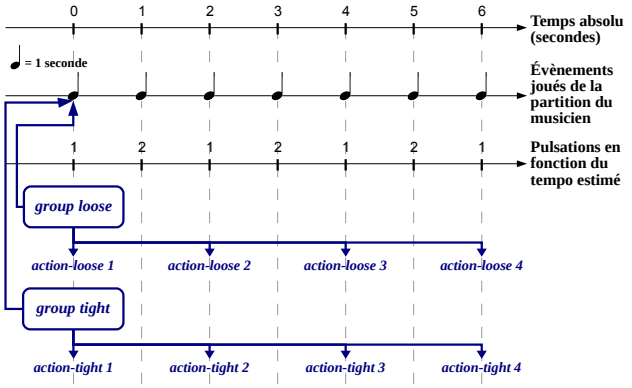
$$synchro = \texttt{loose} \mid \texttt{tight}\,.$$

2.2.2. *Gestion des erreurs*

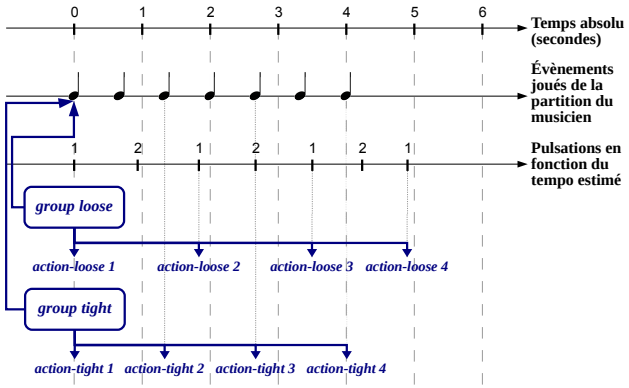
Malgré les très bonnes performances de l'algorithme de reconnaissance, différents cas d'erreur peuvent être rencontrés : l'ordinateur peut confondre un événement avec un autre, ne pas détecter un événement, confondre un non-événement avec un événement. Il faut également considérer les erreurs possibles 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.

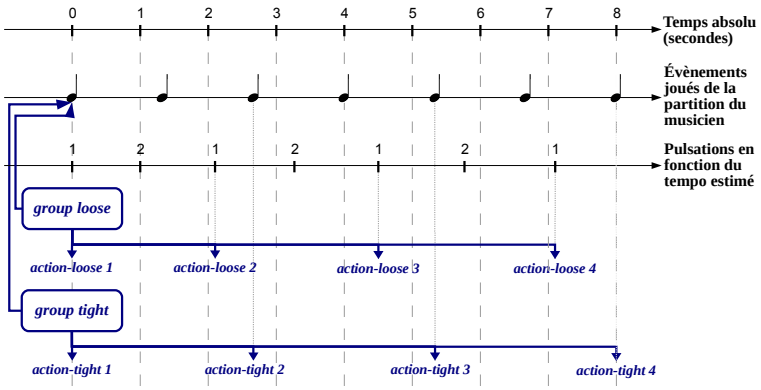
On offre la possibilité au compositeur de spécifier si une action est *globale* (attribut `global`) ou *locale* (attribut `local`) à un événement. L'idée est qu'une action `global` sera lancée (éventuellement en retard) même si l'événement associé n'est pas



(a) interprétation idéale



(b) interprétation rapide



(c) interprétation lente

Figure 2.1 – Exemple d'une même partition interprétée de 3 façons différentes. Les actions des groupes `loose` suivent le *tempo estimé* et donc nécessite un certain temps pour s'adapter si le musicien ne respecte pas exactement le *tempo spécifié*. En revanche, les actions des groupes `tight` sont synchronisés avec les événements dans tous les cas.

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 :

$$error = \texttt{global} \mid \texttt{local}\,.$$

2.2.3. *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 2.2).

– `local` - `loose` : si l'événement associé au début du bloc ($e1$ sur la figure 2.2) 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.

2.2.4. *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.

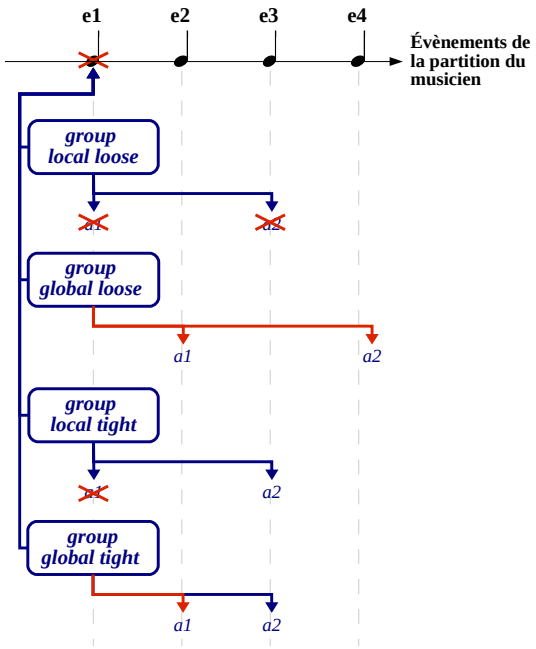


Figure 2.2 – Comportements possibles des blocs d'actions en cas d'erreur

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.

2.3. Datation des actions

Pour formaliser la sémantique de la partition augmentée, nous allons construire un ensemble d'actions et événements symboliquement datés grâce aux fonctions ***datation*** et ***date*** (cf. figure 2.3). Deux autres ensembles ordonnancés pourront ensuite en être déduits.

– Le premier correspondra à la performance idéale de la partition, à savoir sans erreurs et en calculant les dates des actions et événements à partir des tempi spécifiés. On utilisera pour cela les fonctions evt_evalI et act_evalI (cf. figure 2.3).

– Le deuxième sera associé à une performance réelle. Les actions seront datées à partir des dates réelles des événements, des erreurs du système et des différents tempi estimés. On utilisera la fonction act_evalR (cf. figure 2.3).

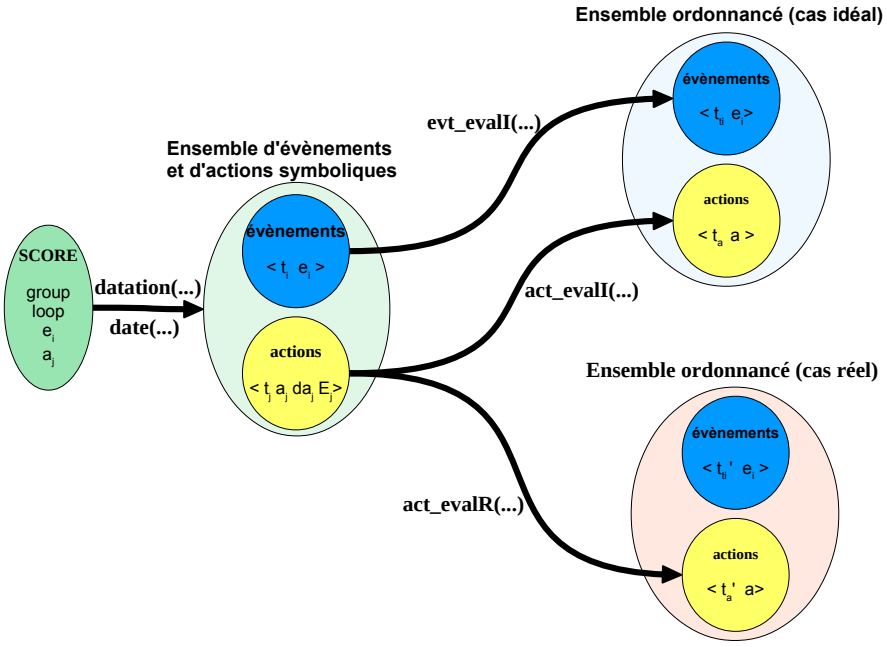


Figure 2.3 – Présentation des différents ensembles et fonctions concernés pour la datation des actions

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.

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

2.3.1. *Datation des événements et des actions*

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 :

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

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

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

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

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

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

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

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

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 ***datation***$[\![\cdot]\!]$ et ***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.

2.3.1.1. *Trace dans la cas idéal*

On calcule les dates de tous les événements et de toutes les actions élémentaires obtenus 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 :

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

2.3.1.2. *Trace dans le cas d'une performance réelle*

Au cours d'une performance, les événements détectés $\mathcal{E}_R = \{e_1^{'}, \ldots, e_n^{'}\}$ sont associés aux dates réelles $\mathcal{T}_R = \{t_1^{'}, \ldots, t_n^{'}\}$. L'indice R correspondant à « réel ». 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 !).

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

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

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 $t_i^{'}$.

Remarquons que 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.

2.3.2. *Exemples de datations*

Nous allons illustrer la datation des traces et le traitement des erreurs, à partir d'un exemple minimal correspondant à une partition $score$ comportant 4 notes et un groupe comportant deux actions (cf figure 2.4) :

$$score = e_1\ (0\ \texttt{G}\ synchro\ error\ (0\ a_1)\ (2\ a_2))\ e_2\ e_3\ e_4$$

La datation de cette partition se calcule comme suit :

$$\begin{aligned} \textbf{\textit{datation}}[\![score]\!] = \textbf{\textit{datation}}[\![e_1]\!] \ \cup\ \textbf{\textit{datation}}[\![0\ \texttt{G}\ \texttt{loose}\ \texttt{local}\ (0\ a_1)\ (2\ a_2)]\!] \\ \cup\ \textbf{\textit{datation}}[\![e_2]\!] \ \cup\ \textbf{\textit{datation}}[\![e_3]\!] \ \cup\ \textbf{\textit{datation}}[\![e_4]\!] \end{aligned}$$

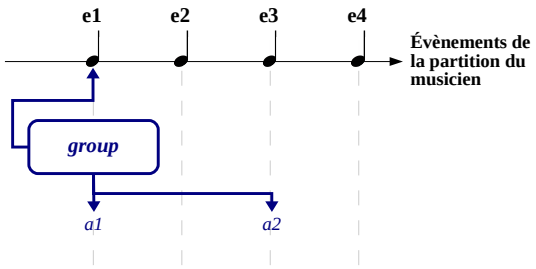


Figure 2.4 – Événements et actions correspondant à l'exemple traité dans 2.3.2

On dérive directement les dates des événements (qui sont indépendantes des attributs du groupe) :

datation$[\![e_1]\!] = \{\langle 0, e_1\rangle\}$; ***datation***$[\![e_2]\!] = \{\langle 1, e_2\rangle\}$;
datation$[\![e_3]\!] = \{\langle 2, e_3\rangle\}$; ***datation***$[\![e_4]\!] = \{\langle 3, e_4\rangle\}$.

Dans la suite, on suppose une même interprétation spécifiée par les événements reconnus et les dates réelles suivants : $\mathcal{E}_R = \{e_2^{'}, e_3^{'}, e_4^{'}\}$; $\mathcal{E}_{Err} = \{e_1^{'}\}$ et $\mathcal{T}_R = \{t_2^{'} = 1.1, t_3^{'} = 2.2, t_4^{'} = 3.3\}$.

1er cas : $synchro =$ `loose` ***et*** $error =$ `local`

On dérive la datation du groupe :

$$
\begin{aligned}
&\textbf{\textit{datation}}[\![1\ \texttt{G loose local}\ (0\ a_1)\ (2\ a_2)]\!] \\
&\quad = \textbf{\textit{date}}_{(1,\varnothing)}[\![0\ \texttt{G loose local}\ (0\ a_1)(2\ a_2)]\!] \\
&\quad = \textbf{\textit{date}}_{(1,\{\langle e_1,\texttt{local},0\rangle\})}[\![0\ a_1]\!] \ \cup\ \textbf{\textit{date}}_{(1,\{\langle e_1,\texttt{local},2\rangle\})}[\![2\ a_2]\!] \\
&\quad = \{\langle 1, a_1, 0, \{\langle e_1, \texttt{local}, 0\rangle\}\rangle,\ \ \langle 1, a_2, 2, \{\langle e_1, \texttt{local}, 2\rangle\}\rangle\}
\end{aligned}
$$

et la datation de la trace correspondante :

$$
\begin{aligned}
&evalR_A(\langle 1, a_1, 0, \{\langle e_1, \texttt{local}, 0\rangle\}\rangle = \bot \\
&evalR_A(\langle 1, a_2, 2, \{\langle e_1, \texttt{local}, 2\rangle\}\rangle) = \bot
\end{aligned}
$$

2ème cas : $synchro =$ `loose` ***et*** $error =$ `global`

$$
\begin{aligned}
&\textbf{\textit{datation}}[\![1\ \texttt{G loose global}\ (0\ a_1)\ (2\ a_2)]\!] \\
&\quad = \textbf{\textit{date}}_{(1,\varnothing)}[\![0\ \texttt{G loose global}\ (0\ a_1)(2\ a_2)]\!] \\
&\quad = \textbf{\textit{date}}_{(1,\{\langle e_1,\texttt{global},0\rangle\})}[\![0\ a_1]\!] \ \cup\ \textbf{\textit{date}}_{(1,\{\langle e_1,\texttt{global},2\rangle\})}[\![2\ a_2]\!] \\
&\quad = \{\langle 1, a_1, 0, \{\langle e_1, \texttt{global}, 0\rangle\}\rangle,\ \ \langle 1, a_2, 2, \{\langle e_1, \texttt{global}, 2\rangle\}\rangle\}
\end{aligned}
$$

et la datation de la trace correspondante :

$evalR_A(\langle 1, a_1, 0, \{\langle e_1, \texttt{global}, 0\rangle\}\rangle = \ 1.1$
$evalR_A(\langle 1, a_2, 2, \{\langle e_1, \texttt{global}, 2\rangle\}) = \ 3.1$

3ème cas : $synchro = \texttt{tight}$ ***et*** $error = \texttt{local}$

datation $[\![0\ \texttt{G tight local}\ (0\ a_1)\ (2\ a_2)]\!]$
$= \textbf{\textit{date}}_{(1,\varnothing)}[\![0\ \texttt{G tight local}\ (0\ a_1)(2\ a_2)]\!]$
$= \textbf{\textit{date}}_{(1,\{\langle e_1,\texttt{local},0\rangle\})}[\![0\ a_1]\!] \ \cup\ \textbf{\textit{date}}_{(1,\varnothing)}[\![0\ \texttt{G tight local}\ (2\ a_2)]\!]$
$= \{\langle 1, a_1, 0, \{\langle e_1, \texttt{local}, 0\rangle\}\rangle\} \ \cup\ \textbf{\textit{date}}_{(2,\varnothing)}[\![0\ \texttt{G tight local}\ (1\ a_2)]\!]$
$= \{\langle 1, a_1, 0, \{\langle e_1, \texttt{local}, 0\rangle\}\rangle\} \ \cup\ \textbf{\textit{date}}_{(3,\varnothing)}[\![0\ \texttt{G tight local}\ (0\ a_2)]\!]$
$= \{\langle 1, a_1, 0, \{\langle e_1, \texttt{local}, 0\rangle\}\rangle\} \ \cup\ \textbf{\textit{date}}_{(3,\{\langle e_3,\texttt{local},0\rangle\})}[\![0\ a_2]\!]$
$= \{\langle 1, a_1, 0, \{\langle e_1, \texttt{local}, 0\rangle\}\rangle \ \ \langle 3, a_2, 0, \{\langle e_3, \texttt{local}, 0\rangle\}\rangle\}$

$evalR_A(\langle 1, a_1, 0, \{\langle e_1, \texttt{local}, 0\rangle\}\rangle = \perp$
$evalR_A(\langle 3, a_2, 0, \{\langle e_3, \texttt{local}, 0\rangle\}) = 2.2$

4ème cas : $synchro = \texttt{tight}$ ***et*** $error = \texttt{global}$

datation $[\![0\ \texttt{G tight global}\ (0\ a_1)\ (2\ a_2)]\!]$
$= \textbf{\textit{date}}_{(1,\varnothing)}[\![0\ \texttt{G tight global}\ (0\ a_1)(2\ a_2)]\!]$
$= \textbf{\textit{date}}_{(1,\{\langle e_1,\texttt{global},0\rangle\})}[\![0\ a_1]\!] \ \cup\ \textbf{\textit{date}}_{(1,\varnothing)}[\![0\ \texttt{G tight global}\ (2\ a_2)]\!]$
$= \{\langle 1, a_1, 0, \{\langle e_1, \texttt{global}, 0\rangle\}\rangle \ \ \langle 3, a_2, 0, \{\langle e_3, \texttt{global}, 0\rangle\}\rangle\}$

$evalR_A(\langle 1, a_1, 0, \{\langle e_1, \texttt{global}, 0\rangle\}\rangle = 1.1$
$evalR_A(\langle 3, a_2, 0, \{\langle e_3, \texttt{global}, 0\rangle\}) = 2.2$

2.4. Gestion des variables

Une part importante du travail effectué lors de ce stage a porté sur l'intégration des variables dans le système Antescofo. Cet enrichissement du langage découle d'une demande des compositeurs. En effet, un certain nombre d'entre eux, utilisateurs réguliers d'Antescofo, ont ressenti le besoin de spécifier les processus électroniques en fonction de paramètres dépendants de la performance (en plus du tempo déjà accessible) et pouvant évoluer dynamiquement. Les variables rendent possibles la description de tels processus. Elles permettent ainsi d'augmenter les interactions et corrélation possibles entre interprétation et actions générées.

Un autre avantage réside dans le fait que les variables peuvent simplifier considérablement la partition. En effet, il arrive souvent que l'évolution temporelle d'un processus musical émane d'une idée « algorithmique », pouvant être décrite succinctement à l'aide de variables. L'écriture du temps est alors plus intuitive et plus proche de l'idée compositionnelle de départ. Les variables facilitent la lisibilité et la modification de la partition et rendent possible l'abstraction.

2.4.1. *Stratégie mise en place*

L'implantation effectuée est un prototype ; en voici les principales caractéristiques.

Toutes les déclarations, les écritures et les lectures de variables s'effectuent à l'intérieur des blocs d'actions. La lecture est autorisée au sein de n'importe quel bloc. L'écriture n'est permise qu'au niveau du bloc où la variable a été déclarée. Au moment de la déclaration d'une variable, le bloc devient en quelque sorte, propriétaire de celle-ci. On évite ainsi d'avoir à gérer les accès concurrents à l'aide de sémaphores et de priorités qui pourrait entraîner des attentes et/ou des interblocages tout en assurant une cohérence maximale du système.

Les variables n'ont pour l'instant qu'un seul type possible : les flottants.

Toutes les variables sont précédés du signe « $ » imitant ainsi les conventions de Max/MSP, Pure Data et PHP.

Les écritures sont équivalentes à des actions élémentaires à l'intérieur d'un bloc. Elles apparaissent au même niveau, précédés d'un certain délai ; la gestion temporelle de l'écriture effective correspond tout à fait à celle de l'exécution d'une action. Quand une suite d'écritures successives est spécifiée avec des délais de 0 entre les écritures, les affectations de variables sont réalisées dans l'ordre de leur spécification.

Les lectures sont autorisées au niveau des délais précédant les actions, des périodes associées aux $loop$ et à l'intérieur de n'importe quelle action élémentaire.

2.4.2. *Stratégies d'évaluations des variables*

L'introduction des variables rend plus visible la différence entre deux modes d'évaluation des messages. Il est naturel de distinguer le moment de création du message, de celui de son envoi effectif, après écoulement de son délai. On peut donc évaluer ses arguments soit à la création (liaison précoce ou « early binding » en anglais), soit à son envoi (liaison tardive ou « late binding »). Ces deux stratégies d'évaluation produisent les mêmes résultats quand les arguments d'un message sont tous des constantes. Par contre, pour le message :

```
4.0 msg($v)
```

les deux stratégies conduisent à un résultat différent si la valeur de la variable `$v` est modifiée entre la date t de création de ce message et la date $t+4$.

Actuellement, c'est la liaison tardive qui est utilisée dans la stratégie d'évaluation. Nous envisageons cependant l'introduction d'autres stratégies d'évaluation et nous étudions différentes « vies » (statique, dynamique, durant la phase compositionnelle), et durées de vies possibles des variables. En effet, l'analyse des cas d'utilisation des variables montre une grande richesse de comportements possibles.

Par exemple, lorsqu'une variable est utilisée comme délai ou période précédant une action a, elle est dans la version actuelle, évaluée au moment où l'on entame le

décompte du délai. Si sa valeur est modifiée avant que ce décompte ne soit terminé, il n'y a, dans l'état actuel du code, pas de ré-évaluation. Cependant le comportement inverse qui consisterait donc à ré-évaluer la durée du délai en fonction des variations de la variable en cours de décompte, présente un certain intérêt. Il permettrait par exemple d'assouplir les contraintes de dépendances entre variables de deux groupes. Il existe déjà dans le système un mécanisme semblable de ré-évaluation des durées en cas de changement de tempo. Par exemple, lors d'une exécution, le système calcule d'abord le délai réel d'une action, par rapport à la durée relative spécifiée et au tempo courant estimé ; si le tempo change avant que l'action ne soit lancée, alors le délai réel restant est recalculé en fonction du temps réel écoulé, de la durée relative et des deux valeurs du tempo. On pourrait combiner les deux mécanismes de ré-évaluation dynamique. Nous proposons d'utiliser un patron de conception observateur/observable pour ce faire. Ce travail est en cours d'implantation. Différentes stratégies similaires au rattrapage d'erreurs pourraient décider du lancement ou non de l'action a, au cas où le résultat de la ré-évaluation du délai soit négatif, c'est-à-dire lorsque la nouvelle date de lancement de l'action correspond à un instant passé.

2.5. Développement

2.5.1. *Architecture du système Antescofo*

Antescofo a été développé comme un module externe des environnements de programmation Max/MSP et PureData. Le langage de programmation est le C++ et la librairie *flext* (Grill, 2004) est utilisée pour faciliter l'intégration avec Max/MSP et Pure Data.

L'implantation profite de nombreuses fonctionnalités offertes par le paradigme de programmation orientée objet, notamment la généricité (templates) et les patrons de conception. Le code est structuré autour d'un fichier source `antescofo.cpp` et d'une vingtaine de fichier d'en-tête correpondant à différentes classes (cf. figures 2.6 et 2.5). Voici un bref descriptif des principales classes :

antescofo : C'est la classe principale. On y définit les fonctions accessibles via l'interface utilisateur (chargement de la partition, démarrage, alignement au niveau de l'action suivante, etc.), les instanciation de quelques unes des principales classes et les différentes relations qu'elle partagent.

AnteParser : Classe effectuant l'analyse syntaxique de la partition. Elle initialise la plupart des structures de données nécessaires lors de l'exécution (timers, t_fwd, t_event, variables).

AnteTimer : C'est au niveau de cette classe que sont gérés le lancement des actions, la synchronisation, le traitement des erreurs. On associe à chaque partition, un « timer » principal, instance de cette classe. Un de ses attributs correspond à la liste des instances de la classes t_fwd associés aux actions du premier niveau

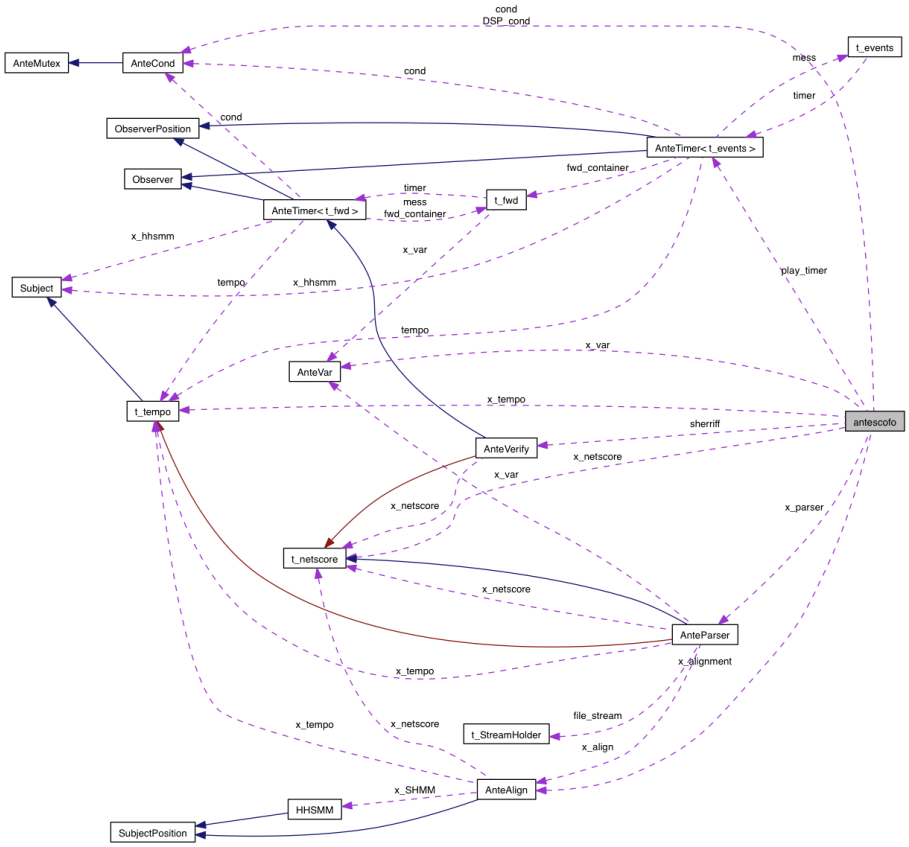


Figure 2.5 – organisation et dépendances des classes d'Antescofo

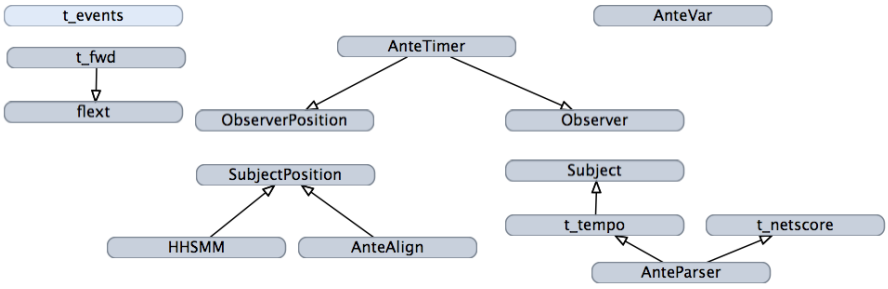


Figure 2.6 – organisation des principales classes d'Antescofo

de la hiérarchie (directement associés à un événement) auxquelles sont associés leurs propres « timer ». Un élément de cette liste peut correspondre à un groupe d'actions auxquel on va associer un « timer », contenant un ou plusieurs t_fwd et ainsi de suite. La figure 2.7 nous montre un exemple de hiérarchisation des actions et de leur correspondance au niveau du code. Un timer est activé (par le système Max/MSP ou Pure Data sous-jacent) et exécute alors les actions qui lui sont associées.

AnteAlign : Classe correspondant à l'algorithme de reconnaissance. Elle notifie aux instances abonnées de AnteTimers, tout changement de la position courante (patron de conception Publication/Inscription).

ScoreClass : Classe modélisant la partition avec l'ensemble des événements et actions.

AnteTempo : Classe correspondant à l'algorithme d'estimation du tempo. Au cours de l'exécution les changements de tempi sont notifiés aux instances de AnteTimer abonnées.

t_fwd : Une instance de cette classe correspond à un groupe d'actions ou à une action élémentaire. Un des principaux attributs est le timer associé à chaque action. On y gère entre autres choses la communication avec la classe des variables.

eventClass : Classe modélisant les événements à reconnaître.

AnteVar : Classe pour la gestion des variables

2.5.2. *Exemple d'exécution*

La figure 2.8 reprend l'exemple de la figure 2.7, pour montrer l'interaction entre les différentes classes, du chargement de la partition jusqu'aux déclenchements des actions, en tenant compte notamment de la hiérarchie des groupes.

2.6. Validation

Afin de tester et évaluer l'ensemble du travail effectué lors de ce stage, nous avons choisi des exemples musicaux réels. Nous décrirons dans cette section les apports des nouvelles fonctionnalités du langage sur quelques uns d'entre-eux.

2.6.1. *Stratégies de synchronisation*

2.6.1.1. *New York Counterpoint (movement 3) de Steve Reich*

La figure 2.9 permet de visualiser les deux stratégies de synchronisation $loose$ et $tight$. Les deux sous-figures correspondent à un même extrait d'une oeuvre de Steve Reich pour clarinette, le troisième mouvement de New York Counterpoint. Lors de

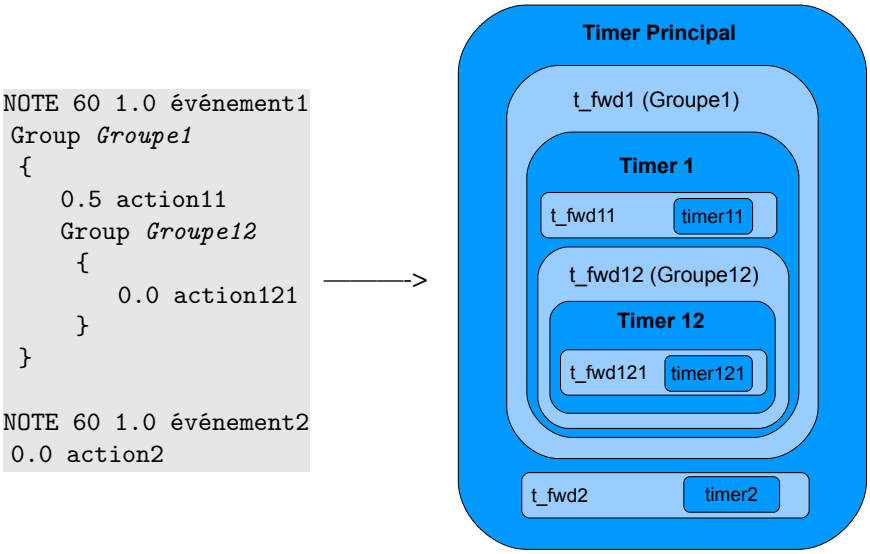


Figure 2.7 – exemple d'actions à deux niveaux de hiérarchie avec leurs correspondances au niveau de l'implémentation

l'enregistrement, il avait été demandé au musicien d'accélérer et de ralentir à sa guise pour tester le système Antescofo. Une boucle *loop* composée d'une seule action joue un rôle de métronome adaptatif ; les marqueurs verticaux de la figure correspondent à la pulsation de ce processus. Dans le premier cas (figure 2.9a), la boucle a une stratégie de synchronisation *loose*. Temporellement elle ne s'appuie donc que sur l'estimation du tempo. Dans le second cas (figure 2.9b), on associe la stratégie *tight* à la boucle. Dans l'extrait choisi, les notes jouées par le musicien ne correspondent qu'à des temps ou des contre-temps. On remarque que les marqueurs correspondant à la stratégie *tight* sont synchrones avec le musicien ; les marqueurs correspondant à la stratégie *loose* ne tombent pas avec les événements, mais ils conservent une continuité malgré les « à-coups » du musicien.

2.6.1.2. *Sonate n°4 de Haendel pour flûte et clavecin*

Dans un contexte d'accompagnement automatique, l'ordinateur joue le rôle d'un musicien virtuel et agit en fonction du jeu de l'instrumentiste en temps réel. On délègue les parties qui ne sont pas jouées par le soliste à des synthétiseurs électroniques, à travers les actions du langage. Pour illustrer les stratégies de synchronisation dans ce contexte, on compare l'exécution d'une partition par un ordinateur utilisant un synthétiseur MIDI et l'interprétation de la même partition par un musicien amateur avec

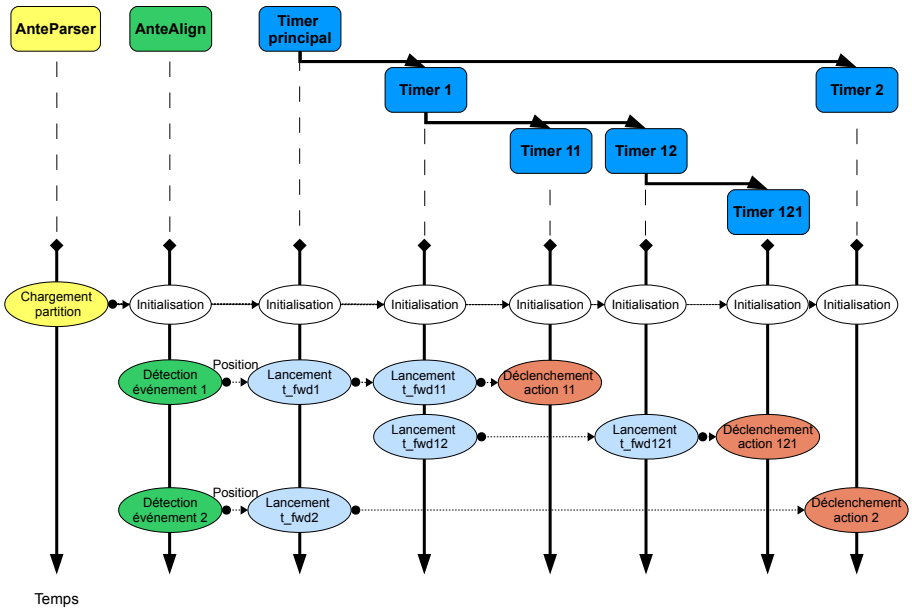


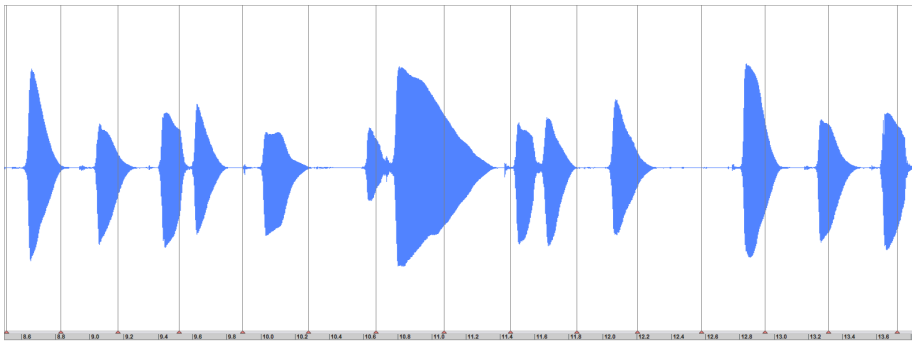
Figure 2.8 – Diagramme de séquence simplifié modélisant l'interaction entre les différentes classes d'Antescofo pendant une exécution

des accompagnements générés par Antescofo utilisant des stratégies différentes (cf. figure 2.10).

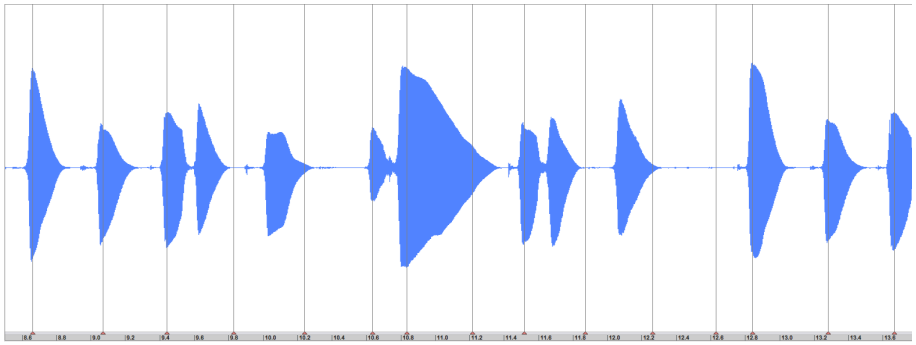
On constate que le schéma global de l'ordonnancement des événements et des actions reste le même, mais leur proportion temporelle ainsi que leurs relations internes fluctuent lors de l'exécution provenant du musicien et de son interprétation. L'utilisation du groupe $loose$ est moins approprié dans ce cadre là ; on remarque en effet que l'accompagnement prend du retard dès les premières notes (cf. figure 2.10b), alors que l'utilisation du groupe $tight$ permet de maintenir à tout moment une synchronisation satisfaisante.

La figure 2.10d met en avant le fait que les actions ne sont pas jouées lorsque elles sont associées à des événements ratés (attribut $local$). De plus, on remarque qu'il n'y a pas de latence au niveau de l'accompagnement, quand la note suivante est jouée.

Nous l'avons vu, le choix d'une stratégie de synchronisation a des conséquences notables durant la performance temps réel. Mais une stratégie de synchronisation peut aussi être considérée comme idée compositionnelle au moment de l'écriture. Par exemple l'oeuvre Einspielung 1, pour violon et électronique, composée par Emmanuel



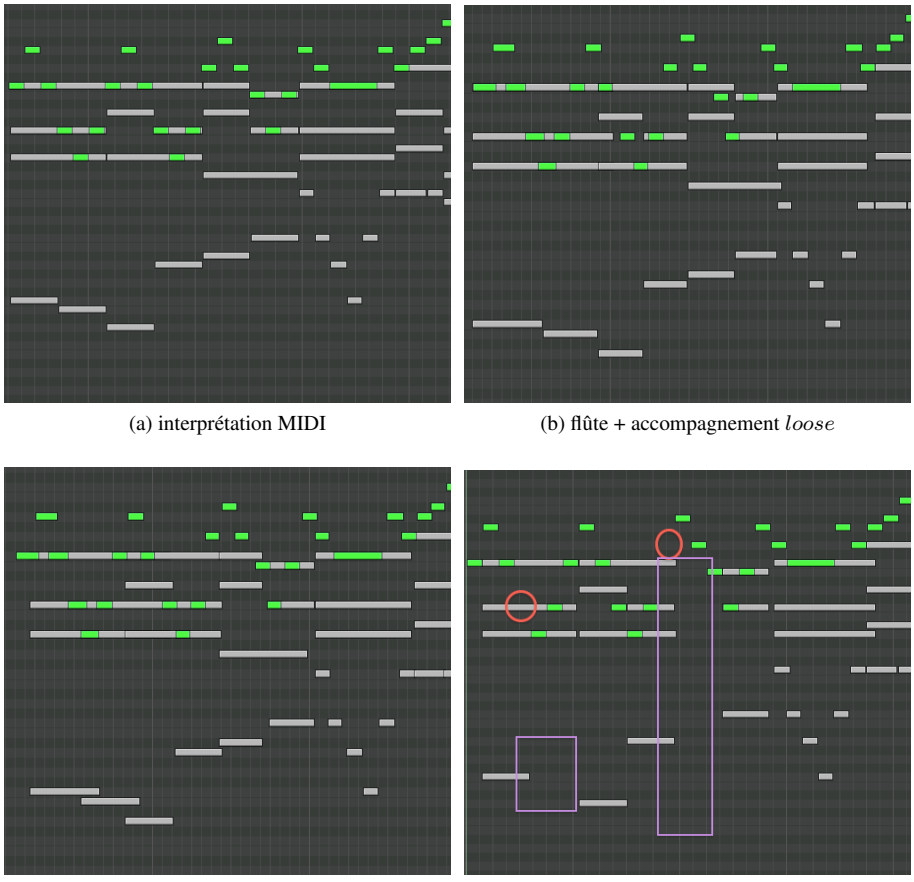
(a) loose



(b) tight

Figure 2.9 – Représentation temporelle d'un enregistrement extrait du troisième mouvement de New York Counterpoint pour clarinette, composé par Steve Reich. Les marqueurs ont été générés en temps réel avec Antescofo et permettent de visualiser les temps dans une boucle $loose$ (2.9a) et $tight$ (2.9b).

Nunes et créée pendant le festival Agora 2011, est un exemple approprié pour l'illustration de l'idée « $tight$ ». En effet, on trouve tout au long de cette oeuvre des traits de notes rapides où le compositeur a associé des actions à chaque événement. Ces actions peuvent correspondre à des harmoniseurs, des ouvertures de filtres, des commandes de spatialisation, etc. L'écriture de l'électronique a été pensée sous la forme de phrases musicales ayant leur propre sens, mais dont chaque élément doit être synchronisé par rapport à un événement capté.



(a) interprétation MIDI

(b) flûte + accompagnement $loose$

(c) flûte + accompagnement $tight$

(d) mélodie avec erreur + accompagnement $local$

Figure 2.10 – Quatre transcriptions (sous forme de piano-roll) d'un extrait de la sonate N°4 de Haendel pour flûte (vert) et clavecin (gris). La figure 2.10a correspond à une interprétation faite par un ordinateur (synthèse MIDI). Les figures 2.10b et 2.10c correspondent à la transcription de l'interprétation (par un flûtiste amateur) de la partition avec l'accompagnement généré automatiquement. L'accompagnement de 2.10b est un groupe $loose$; l'accopagnement de 2.10c est un groupe $tight$. La figure 2.10d correspond à une interprétation dans laquelle on a introduit deux erreurs (en rouge) dans la mélodie ; l'accompagnement $tight$ et $local$ réagit en conséquence (les cadres violets signalent l'accompagnement qui n'est pas joué)

2.6.2. *Variables*

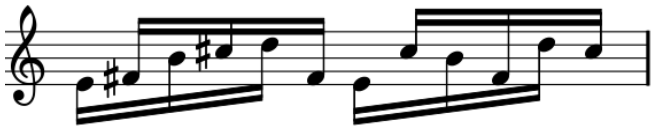
2.6.2.1. *Pianophases de Steve Reich*

Le compositeur Steve Reich est considéré comme l'un des pionniers de la musique minimaliste. Il s'est notamment fait connaître par ses musiques de phases. Le principe de la musique de phase, consiste à faire jouer le même motif cyclique par deux voix différentes. Une des deux voix va accélérer légèrement à des instants périodiques, se décaler, jusqu'à qu'elle se stabilise avec la deuxième voix, en formant une nouvelle superposition entre elles. Ce processus se répète jusqu'à que les deux voix jouent le motif à l'unisson, comme au début.

L'exemple de la figure 2.11 nous montre un de ces motifs et son écriture dans le langage d'actions d'Antescofo. Les deux boucles décrivent le même motif. La deuxième boucle diminue période et délais toutes les 8 périodes jusqu'à se décaler d'une double-croche par rapport à la boucle référence, puis restitue période et délai à leur valeur initiale. On alterne ainsi une phase de décalage (4 périodes), avec une phase de stabilité (4 périodes). Sans les variables, l'écriture de l'évolution temporelle de ce processus musical serait très fastidieuse ; il faudrait écrire un cycle correspondant à la première voix, un cycle correspondant à la deuxième voix pour les 12 phases de stabilité, et 4 cycles pour les 12 phases de transition soit (1+13+4*12)*12 = 744 actions à écrire.

2.6.2.2. *HistWhist de Marco Stroppa*

Le compositerur italien Marco Stroppa est un utilisateur régulier d'Antescofo et participe activement à son évolution. Dans son oeuvre intitulé Hist whist (2009) pour violon et électronique, il utilise le système Antescofo et en particulier son langage pour décrire la partition. La figure 2.12a en est un extrait simplifié. Il s'agit d'un processus électronique composé de deux actions lancées avec un certain délai. Ce processus boucle mais sa période augmente à chaque pas, jusqu'au dixième cycle. A partir du 11ème cycle, la période reste stable et c'est le délai entre les deux actions qui augmente à chaque pas. Le compositeur utilise pour cela une série d'appels à des macros qui créent puis détruisent le processus électronique, en donnant en argument les valeurs adéquates. Cette technique est astucieuse mais difficile à mettre en place, à comprendre et donc à modifier. La figure 2.12b spécifie le même processus musicale mais en utilisant cette fois les variables. La simplification de l'écriture dans cet exemple, nous montre l'intérêt des variables pour la composition de processus musicaux qui évoluent dynamiquement tout en préservant une base structurelle.



```
Loop LoopRéf 3.0
{  0.00 mnote 64 64 @beat2ms(0.25)
   0.25 mnote 66 64 @beat2ms(0.25)
   0.25 mnote 71 64 @beat2ms(0.25)
   0.25 mnote 73 64 @beat2ms(0.25)
   0.25 mnote 74 64 @beat2ms(0.25)
   0.25 mnote 66 64 @beat2ms(0.25)
   0.25 mnote 64 64 @beat2ms(0.25)
   0.25 mnote 73 64 @beat2ms(0.25)
   0.25 mnote 71 64 @beat2ms(0.25)
   0.25 mnote 66 64 @beat2ms(0.25)
   0.25 mnote 74 64 @beat2ms(0.25)
   0.25 mnote 73 64 @beat2ms(0.25) }

Loop LoopDec $période
    ($période=3.0) ($délai=0.25) ($compteur=0)
{  0.0 mnote 64 64 @beat2ms(0.25)
   $délai mnote 66 64 @beat2ms(0.25)
   $délai mnote 71 64 @beat2ms(0.25)
   $délai mnote 73 64 @beat2ms(0.25)
   $délai mnote 74 64 @beat2ms(0.25)
   $délai mnote 66 64 @beat2ms(0.25)
   $délai mnote 64 64 @beat2ms(0.25)
   $délai mnote 73 64 @beat2ms(0.25)
   $délai mnote 71 64 @beat2ms(0.25)
   $délai mnote 66 64 @beat2ms(0.25)
   $délai mnote 74 64 @beat2ms(0.25)
   $délai mnote 73 64 @beat2ms(0.25)
   $compteur = ( $compteur +1) % 8
   0.0 $période = if($compteur = 4, $période - 0.25 / 4, $période)
   0.0 $période = if ($compteur = 0, $période + 0.25 / 4, $période)
   0.0 $délai = $période / 12}
```

Figure 2.11 – Motif extrait de l'oeuvre « Piano Phase » de Steve Reich et de sa description dans le langage d'Antescofo.

```
@MACRO_DEF harm_gain( NUM, INIT_DELAY, pnum , PERIOD, INTERNAL_PER, LOWG, HIGHG )
{
    LFWD    INIT_DELAY hNUM-PERIOD-INTERNAL_PER_p   PERIOD
    {
        harmNUM HIGHG   @beat2msec( INTERNAL_PER )  @name hNUM_p_1
        INTERNAL_PER    harmNUM LOWG    @beat2msec( INTERNAL_PER )  @name hNUM_p_2
    }
    print NUM gain global PERIOD local INTERNAL_PER
    KILL expr{ pnum*PERIOD - PERIOD/50.0}   hNUM-PERIOD-INTERNAL_PER_p
}


@MACRO_DEF harm_hob_gain( NUM, INIT_PERIOD, FINAL_PERIOD, INTERNAL_PERIOD, INTERNAL_P_FIN )
{
    GFWD harm_nobNUM
    {
        @harm_gain(NUM, 0.0, 1.0, INIT_PERIOD, INTERNAL_PERIOD, -30.0, 3.0 )
        @harm_gain( NUM, expr{INIT_PERIOD/50.0}, 1.0, 1, 1, -30.0, 3.0 )
        @harm_gain( NUM, expr{INIT_PERIOD/50.0}, 1.0, 2, 1, -30.0, 3.0 )
        @harm_gain( NUM, expr{INIT_PERIOD/50.0}, 1.0, 3, 1, -30.0, 3.0 )
        @harm_gain( NUM, expr{INIT_PERIOD/50.0}, 1.0, 4, 1, -30.0, 3.0 )
        @harm_gain( NUM, expr{INIT_PERIOD/50.0}, 1.0, 5, 1, -30.0, 3.0 )
        @harm_gain( NUM, expr{INIT_PERIOD/50.0}, 1.0, 6, 1, -30.0, 3.0 )
        @harm_gain( NUM, expr{INIT_PERIOD/50.0}, 1.0, 7, 1, -30.0, 3.0 )
        @harm_gain( NUM, expr{INIT_PERIOD/50.0}, 1.0, 8, 1, -30.0, 3.0 )
        @harm_gain( NUM, expr{INIT_PERIOD/50.0}, 1.0, 9, 1, -30.0, 3.0 )
        @harm_gain( NUM, expr{INIT_PERIOD/50.0}, 1.0, 10, 1, -30.0, 3.0 )

        @harm_gain( NUM, expr{INIT_PERIOD/50.0}, 1.0, 10, 1.5, -30.0, 3.0 )
        @harm_gain( NUM, expr{INIT_PERIOD/50.0}, 1.0, 10, 2.0, -30.0, 3.0 )
        @harm_gain( NUM, expr{INIT_PERIOD/50.0}, 1.0, 10, 2.5, -30.0, 3.0 )
        @harm_gain( NUM, expr{INIT_PERIOD/50.0}, 1.0, 10, 3.0, -30.0, 3.0 )
        @harm_gain( NUM, expr{INIT_PERIOD/50.0}, 1.0, 10, 3.5, -30.0, 3.0 )
    }
}

BPM 60.0
NOTE 60 10.0    m2_e2064

    @harm_hob_gain(1, 0.4, 1.6, 0.2, 0.8)
```

(a) sans les variables

```
BPM 60.0
NOTE 60 10.0    m2_e2064


    LFWD processus_name $PERIOD
         ($INIT_DELAY=0.0) ($PERIOD=1) ($INTERNAL_PER=1.0) ($compt=0)

    {
        $INIT_DELAY harm$NUM 3.0    @beat2msec( $INTERNAL_PER ) @name h$NUM_p_1
        INTERNAL_PER    harm$NUM    -30 @beat2msec( $INTERNAL_PER ) @name h$NUM_p_2

        ;GESTION DES VARIABES
        $PERIOD = if( $compt < 10 , $PERIOD = $PERIOD + 1, $PERIOD)
        $INTERNAL_PER = if( $compt > 11 , $INTERNAL_PER = $INTERNAL_PER + 0.5, $INTERNAL_PER)
        $$compt = ( $compt +1)
    }
```

(b) avec les variables

Figure 2.12 – Deux versions du même extrait simplifié de Hist whist composé par Marco Stroppa ; l'utilisation des variables dans le second simplifie l'écriture du processus électronique

Chapitre 3

Conclusion et perspectives

3.1. Conclusion

Le travail réalisé lors de ce stage aborde des problématiques liées à l'accompagnement musical temps réel et à l'écriture de l'interaction musicien/machine dans le cadre du système de suivi de partition Antescofo (`http://repmus.ircam.fr/antescofo`). 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.

D'un point de vue scientifique, le système de reconnaissance d'Antescofo est le résultat d'une combinaison complexe de savoir-faire scientifiques, en particulier dans les thématiques du traitement du signal et de l'apprentissage automatique, qui en fait un des outils les plus performants dans son domaine.

Le langage proposé aux compositeurs et toute la machinerie que cela implique vise à résoudre le problème de rendre composable et interprétable la musique électronique/mixte, avec des libertés et des abstractions similaires à celles engendrées par la partition et les instruments traditionnels.

Ma contribution se situe essentiellement aux niveaux de l'enrichissement du langage et de la formalisation du comportement des processus (événements joués par le musicien et les actions générées) qu'il décrit. Le code associé à ce travail a été testé et validé notamment par le biais d'oeuvres composées par Emmanuel Nuñes et Marco Stroppa.

D'abord, les différentes stratégies de synchronisation mis en place sont intuitives d'un point de vue musical ; elles facilitent le travail du compositeur au moment de la transcription de l'idée compositionnelle.

D'autre part, le caractère critique d'une représentation musicale avec suivi de partition, nous oblige à proposer des stratégies de rattrapages d'erreurs et incite le compositeur à définir la manière dont les actions vont être générées ou non en cas de problème.

Pour que la réalisation de la partition soit déterministe au sens voulu par le compositeur, il fallait que ces différentes stratégies de synchronisations et de gestion des erreurs soient orthogonales et sans ambiguïtés. Chaque combinaison possible a un sens musical et l'ensemble des combinaisons couvre l'ensemble des comportements souhaités.

Nous nous sommes également intéressés de près aux différents mécanismes proposés par les langages synchrones dits généralistes, notamment la gestion des variables dans chacun d'entre eux, la gestion des horloges dans lucid synchrone (Wadge *et al.*, 1985) et Lucy-n (Mandel *et al.*, 2010) ou la notion de signal dans Esterel (Berry *et al.*, 1992).

Les problèmes que nous nous posions étaient cependant trop spécifiques à la musique, pour que ces concepts pré-existants nous satisfassent.

Antescofo est le premier système à présenter un langage dédié à la spécification de l'interaction temps réel entre un musicien et un environnement de programmation musical. C'est pourquoi la comparaison avec les autres langages pour l'informatique musical est délicate car ils sont le plus souvent complémentaires.

Faust (Orlarey *et al.*, 2004) propose un langage à flot de données avec une sémantique synchrone, dédié à la définition d'applications pour le traitement du signal.

SuperCollider (Mccartney, 1996) offre un environnement et un langage de programmation pour la synthèse audio temps réel et la composition algorithmique, capable de compiler du code à la volée.

OpenMusic (Assayag *et al.*, 1999) donne accès à un ensemble d'outils graphique pour les compositeurs. La sémantique de son langage graphique est proche du paradigme synchrone. Le but est de permettre au compositeur de construire son propre environnement de travail mais le système n'a pas de vocation temps réel.

Enfin Antescofo est très lié à Max/MSP puisque le premier est un module du deuxième, même s'ils sont conceptuellement très différents.

3.2. Perspectives

Des extensions, sollicitées par les utilisateurs, sont actuellement à l'étude, comme des groupes pouvant être créés dynamiquement à la reconnaissance d'un certain pattern, ou la possibilité d'exprimer d'autres types de synchronisation, par exemple par contraintes, ce qui permettrait une écriture plus souple pour le compositeur. Une sémantique graphique à la StateCharts (Harel, 1987) est envisagée, en collaboration avec l'éditeur de partition NoteAbility Pro, afin de faciliter le travail d'écriture. Par ailleurs, nous avons adopté un point de vue volontairement dénotationnel pour la formalisation, 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 des 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 interprétation abstraite de systèmes distribués temps-réel avec des horloges désynchronisées.

Ce travail à été l'objet d'une présentation en vu de collaborations futures avec l'équipe Parkas du département d'Informatique de l'École Normale Supérieure. Le colloque de Modélisation des Systèmes Réactifs sera également l'occasion pour nous, d'échanger avec des spécialistes du domaine. J'aurai en effet le privilège de poursuivre ce travail au cours d'une thèse dans l'équipe Représentations Musicales.

Bibliographie

Akshay S., Bollig B., Gastin P., Mukund M., Narayan Kumar K., « Distributed Timed Automata with Independently Evolving Clocks », *in* F. van Breugel, M. Chechik (eds), *Proceedings of the 19th International Conference on Concurrency Theory (CONCUR'08)*, vol. 5201 of *Lecture Notes in Computer Science*, Springer, Toronto, Canada, p. 82-97, August, 2008.

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

André É., Chatain Th., De Smet O., Fribourg L., Ruel S., « Synthèse de contraintes temporisées pour une architecture d'automatisation en réseau », *in* D. Lime, O. H. Roux (eds), *Actes du 7ème Colloque sur la Modélisation des Systèmes Réactifs (MSR'09)*, vol. 43 of *Journal Européen des Systèmes Automatisés*, Hermès, Nantes, France, p. 1049-1064, November, 2009a.

André É., Chatain Th., Encrenaz E., Fribourg L., « An Inverse Method for Parametric Timed Automata », *International Journal of Foundations of Computer Science*, vol. 20, n° 5, p. 819-836, October, 2009b.

Assayag G., Rueda C., Laurson M., Agon C., Delerue O., « Computer Assisted Composition at Ircam : PatchWork & OpenMusic », *Computer Music Journal*, 1999.

Berry G., Gonthier G., « The Esterel Synchronous Programming Language : Design, Semantics, Implementation. », *Sci. Comput. Program.*, vol. 19, n° 2, p. 87-152, 1992.

Bertrane J., « Proving the properties of communicating imperfectly-clocked synchronous systems », *in* K. Yi (ed.), *Proceedings of the Thirteenth International Symposium on Static Analysis (SAS)*, vol. 4134 of *Lecture Notes in Computer Science*, Springer, Seoul, p. 370-386, 29-31 Aug., 2006.

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

Chabot X., Dannenberg R., Bloch G., « A workstation in live performance : Composed improvisation », *International Computer Music Conference (ICMC)*, p. 537-540, Octobre, 1986.

Chadabe J., « Interactive Composing : An Overview », *Computer Music Journal*, vol. 8, n° 1, p. 22-27, 1984.

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 », *Proceedings of International Computer Music Conference (ICMC)*, Belfast, August, 2008.

Cont A., « A coupled duration-focused architecture for realtime music to score alignment », *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, n° 6, p. 974-987, June, 2010.

Dannenberg R. B., « An On-Line Algorithm for Real-Time Accompaniment », *Proceedings of the International Computer Music Conference (ICMC)*, p. 193-198, 1984.

Grill T., *flext C++ programming layer for cross-platform development of PD and Max/MSP externals*, 2004.

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

Harel D., « StateCharts : a Visual Approach to Complex Systems », *Science of Computer Programming*, vol. 8-3, p. 231-275, 1987.

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

Machover T., Chung J., « Hyperinstruments : Musically intelligent and interactive performance and creativity systems », *International Computer Music Conference (ICMC)*, p. 186-190, 1989.

Mandel L., Plateau F., Pouzet M., « Clock Typing of n-Synchronous Programs », *Designing Correct Circuits (DCC 2010)*, Paphos, Cyprus, March, 2010.

Manoury P., *La note et le son*, L'Hamartan, 1990.

Mccartney J., « Supercollider : A new real-time synthesis language », 1996.

Orlarey Y., Fober D., Letz S., « Syntactical and semantical aspects of Faust », *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, vol. 8, p. 623-632, 2004.

Puckette M., « The Patcher », *in* I. C. M. Association (ed.), *International Computer Music Conference (ICMC)*, p. 420-429, 1988.

Puckette M., Lippe C., « Score Following in Practice », *Proceedings of the ICMC*, p. 182-185, 1992.

Raphael C., « The informatics philharmonic », *Commun. ACM*, vol. 54, p. 87-93, March, 2011.

Risset J.-C., « Composing in real-time ? », *Contemporary Music Review*, vol. 18, n° 3, p. 31-39, 1999.

Rowe R., *Interactive music systems : machine listening and composing*, MIT Press, Cambridge, MA, USA, 1992.

Rowe R., *Machine Musicianship*, MIT Press, Cambridge, MA, USA, 2004.

Stroppa M., « Live electronics or live music ? Towards a critique of interaction », *Contemporary Music Review*, vol. 18, n° 3, p. 41-77, 1999.

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

Wadge W., Ashcroft E. A., « Lucid, the Dataflow Programming Language », *Academic Press*, 1985.

Wilhelm R., Engblom J., Ermedahl A., Holsti N., Thesing S., Whalley D., Bernat G., Ferdinand C., Heckmann R., Mitra T., Mueller F., Puaut I., Puschner P., Staschulat J., Stenström P., « 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.

Xenakis, « Vers une métamusique », *La nef*, 1967.