Aspects temporels d'un système de partitions musicales interactives pour la composition et l'exécution

Antoine Allombert

École Doctorale de l'Université Bordeaux 1 **Numéro d'ordre : 3860**

Aspects temporels d'un système de partitions musicales interactives pour la composition et l'exécution

THÈSE

soutenue le 26 octobre 2009

pour l'obtention du

Doctorat de l'Université Bordeaux 1

(spécialité Informatique)

par

Antoine Allombert

Composition du jury

Président :	Robert Strandh - Université Bordeaux 1
Rapporteurs :	Sylviane Schwer - Université Paris 13 Miller Puckette - University of California at San Diego
Examinateurs :	Myriam Desainte-Catherine - Université Bordeaux 1 (directrice) Gérard Assayag - IRCAM-CNRS (co-directeur) Georges Bloch - IRCAM-CNRS

LaBRI – UMR 5800 **IRCAM – UMR 9912**

Cette thèse a été préparée au LaBRI et a l'Ircam.

Remerciements

Je ne saurais débuter ces remerciements sans exprimer ma profonde gratitude à Myriam Desainte-Catherine, pour avoir guidé mes premiers pas dans le monde de la recherche et sans qui rien de tout ceci n'aurait été possible.

Un grand merci à Gérard Assayag pour avoir accepté de co-encadrer ce travail, pour l'accueil au sein de l'équipe *Représentations Musicales*, pour l'ambiance qui y regne, pour les conseils scientifiques et littéaires.

Je remercie l'ensemble des membres du jury.

Au LaBRI... Je remercie l'équipe Son dans son ensemble et tout spécialement Robert Strandh, ainsi que les lispiens du labo. Merci à Aymeric Vincent et Frédéric Herbreteau pour leurs conseils. Une pensée aux membres du projet *Virage*. Un satisfecit reconnaissant à ceux qui, de près ou de loin, ont participé au développement des prototypes : Bruno, Rémi, Délyana... et les étudiants de l'Enseirb. Merci aux membres du *Scrime*, spécialement à Joseph, Gyorgy et Annick.
Sans oublier la folle bande du 308 : Florent, Raphaël et Luc.

A l'Ircam... Merci à Hugues Vinet pour m'avoir permis d'intégrer l'Ircam. Je remercie l'ensemble de l'équipe *Repmus*, et tout particulièrement Jean et Carlos pour leur patience et leur disponibilité, Camilo Rueda pour sa collaboration et Charlotte Truchet pour ses remarques. Merci enfin à quelques frappeurs de volants, à tous ceux qui font que cet institut est ce qu'il est et à Didier Périni, dernier *punk* s'il devait n'en rester qu'un.

En A107... Amitiers très supertitieuses à Grégoire et Yun...

Ailleurs... Mes pensées vont naturellement à ma famille et tout spécialement à mes parents que je remercie affectueusement.
Pensées amicales au Professeur Guiguilele, à Mick, Adrien, Sebastien, la liste Chtenstein et la Vas'compagnie. Salutations, pour leur patience, à Dirty H. et les Hooray Henrys, et pour son indulgence à la bande de *Vivement Mes Vacances*. Une pensée pour la famille Montgolfier et la douce *Malda*. Merci au club cuisine de Bordeaux, aux piscines de St Médard et d'ailleurs... et aux personnes qui vont autour.
A celles et ceux qui ne sont pas là mais à qui je pense...

Merci à Anne de M.

Résumé

Alors que la composition de musique électro-acoustique mobilise de plus en plus d'outils numériques, la question de l'interprétation de telles pièces reste ouverte. La plus part du temps, ces pièces sont un enregistrement sur support, d'une organisation temporel d'un matériau sonore. Pendant l'exécution, l'œuvre est simplement diffusée et l'interprète peut uniquement modifier des paramètres globaux tels que le volume, la balance ou la spatialisation sur le système d'écoute. Il ne peut pas interpréter la pièce au sens où il pourrait le faire pour une partition classique.

Nous souhaiterions que ce type d'interprétation soit possible pour les pièces électro-acoustiques. Cette possibilité ne peut être posible que dans le cadre de "partitions interactives" capables de s'adapter à leur environnement (contrôles de l'interprète, autres musiciens). Dans ce contexte, la partition est exécutée par la machine, ce qui conduit à s'intéresser à un problème différent de celui du suivi de partition.

Nous cherchons à élaborer un système constitué de deux parties : un environnement de composition assistée par ordinateur permettant au compositeur de créer de telles partitions interactives, et une machine d'exécution rendant possible leur interprétation.

L'environnement de composition doit disposer d'une représentation formelle de la musique "interprétable". Nous nous appuyons donc sur une formalisation de l'interprétation de la musique instrumentale, et cherchons alors à la généraliser à des pièces impliquant des processus génériques à la place des notes. Nous focalisons notre étude sur un certain aspet de l'interprétation : les *variations agogiques*, c'est à dire la possibilité pour l'interprète de modifier les date d'occurrence d'événements discrets de la pièces pendant l'exécution. Ces modifications de dates sont accessibles grace à des "point d'interactions" (débuts, fins, ou points intermédiaires) des processus, dont la position temporelle permet de contrôler la temporalité de la pièce.

Mais ces possibilités sont encadrées par le compositeur à la création de la partition (comme dans le cas de musique instrumentale). Ces contraintes sont le résultats d'une démarche de composition, elles permettent également d'éviter une désorganisation totale de la pièce pendant l'exécution.

Nous proposons une représentation formelle des partitions interactives, basées sur des blocs 2D, telle celle des *Maquettes* d'*OpenMusic* ou des *Data Structures* de *Pure Data*. Les partitions sont des ensembles d'objets organisés sur une ligne de temps ; ces objets sont eux-mêmes représentés comme des séquences de points de contrôle discrets (début, fin et points intermédiares). Ils représentent l'exécution de processus responsables du rendu sonore de la pièce (synthèse et traintement de signal ou de symboles ou même opérations algorithmiques complexes).

Pour définir les limites imposées à l'interprétation, le compositeur peut poser des contraintes temporelles entre les points de contrôle de la pièce ; il peut ainsi imposer un ordre partiel entre les événements à l'aide de contraintes qualitatives, ou utiliser des contraintes quantitatives pour limiter les valeurs possibles des intervalles de temps séparant les points de contrôle. La partition est alors à la fois complètement spécifiée, mais sa temporalité reste flexible, laissant ainsi la place à l'interprétation.

De plus, le compositeur peut définir certains points de contrôle comme "interactifs", les rendre ainsi dynamiquement déclenchables à l'arrivée d'événements extérieurs, supposés se produire pendant l'exécution de la pièce. Ces événements peuvent être produits par des interfaces de contrôle, ou par la détection de situations particulières dans le contexte musical.

Lorsqu'une partition est interprétée, la machine d'exécution envoie un message aux processus lorsqu'un point de contrôle doit se produire. Ceci peut être le fait de l'écoulement du temps pour les points non dynamiques, ou de l'arrivée de événement extérieur correspondant pour les points dynamiques. Le système s'assure que les contraintes temporelles définies par le compositeur ne sont pas violées. Si l'arrivée d'un

événement extérieur remet en cause la validité de contraintes, le système recalcule les dates de points de contrôle futurs pour assurer la validité des contraintes. L'ordre entre les points peut alors être modifié. Ces calculs sont effectués par un algorithme de propagation de contraintes. Le maintien des contraintes temporelles, peut amener le système à ignorer des événements extérieurs lorsque ceci risque de contredire des contraintes, tout comme il devra simuler l'arrivée d'un événement dans cas où l'absence de ce dernier conduirait à violer une contrainte.

Nous proposons une struture de machine abstraite capable d'exécuter dynamiquemenr les partitions interactives. Celle-ci est basée sur les réseaux de Petri et la propagation de contraintes. Nous donnons un algorithme pour compiler les partitions depuis leur descritption formelle vers une représentation exécutable par la machine. Nous avons également développé un prototype dans *OpenMusic*, sous la forme d'une extension des *Maquettes* pour l'édition des partitions utilisant un système de propagation de contraintes. Elle contient également un compilateur de partition et une machine d'exécution envoyant des messages UDP vers des applications tierces. Les partitions sont sauvergardées grace à un format XML d'échange. Nous présentons plusieurs applications du système à la musique électro-acoustique la musique instrumentale et le théâtre.

Mots clefs : *Informatique Musicale*, *Relations Temporelles*, *Programmation par Contraintes*, *Interaction*

Abstract

At a time when electro-acoustic music involves more and more signal processing during the composition process, the question of the musical interpretation of such pieces still remains open. Often, these pieces mainly consist in the recording, on a medium, of an in-time organization of "out of time" musical materials. During a performance, the piece is broadcasted while the performer may only modify some global parameters such as the volume, the balance or the spatialisation. But he cannot "interpret" the piece in the same sense that a performer interprets a classical score.

We would like to be able to perform this kind of works within a "musical interpretation" framework. This involves some kind of "interactive score", whose execution may adapt to the environment (e.g. other performers or control interfaces). In such situtation, the score is performed by the machine, leading to a different problematics than the one adressed by score following algorithms.

We aim at creating a system that comprises of : an environment of assisted composition allowing the composers to design interactive scores ; and an execution machine controlled by performers in order to interpret such scores.

We assume that this environment needs a formal representation of "interpretable" music. We base our work on a formalization of the interpretation in instrumental music, and try to generalize it to pieces of music that involve generic processes in place of the "notes". We limit our study to a particular side of the musical interpretation called the "agogic modifications", which means the possibility for the performer to interactively modify the occurence time of some discrete events of the score during the performance. It is important to note that these events are associated with control points (beginnings, ends, or pivot points) of temporal processes, so by controlling these points in time, one may change the temporal deployment of music material.

But these possibilities of "interpretation" are bounded by the composer at the time when the score is composed (just as in instrumental music). Such constraints imposed by the composer may express a compositional strategy. They are also useful for preventing from a disorganization of the piece during the performance.

We propose such a formal description of interactive scores, by using a 2D block representation as one can find in the "Maquettes" of the OpenMusic software or the PD Data Structures . A score is constituted by a set of objects organized over a time-line. These objects are defined as sequences of discrete control points (a beginning, an end and intermediate pivot points). The objects denote the execution of processes that bear the musical content (whether it be signal or event processing or even complex logical computing).

To define the limits of the interpretation frame, the composer can specify temporal constraints between the control points of the score : qualitative precedence constraints used to impose a partial order among the control points ; quantitative constraints used to specify ranges of possible values for the time intervals between control points. The score is thus both completely defined, and flexible in its internal temporal structure, which leaves room to interpretation. In addition the composer may set choosen control points in the score to be" interactive" i.e. linked to external events that are supposed to occur during the performance. These events may consist of control action on an interface or may result from the detection of something happening in the musical context.

When a score is executed, the execution machine tells the processes when the execution time matches the logical time of a control point. This may happen for non-interactive control points just because time has passed, or, in the case of interactive points , because the associed external event has just occured. Simultaneously, the execution machine checks for possible violations of the temporal constraints built into the score. In the case of a violation, the execution machine may change the future mapping between

logical time and execution time in order to maintain score consistancy. This may change the expected execution time for future points, and this may change the ordering as well. A constraint propagator is used for that purpose. Consistance checking may lead the machine to ignore external events in the case when they break the constraint system, or the machine can also "simulate" the occurence of an external event that is awaited for too long and, for that reason, generates a violation in the constraint system.

In addition to the interactive score formalism, we present an abstract machine for the execution that uses an internal representation of the score execution dynamics based on a Petri net associated to a constraints propagation algorithm. We also present an algorithm for compiling the interactive scores from the formal description to the representation used by the execution machine. We have developed a prototype mainly in OpenMusic that comprises : an extension of the OM maquettes for interactive score edition with an XML exchange format and a design constraints propagator ; a compiler ; an execution machine that communicates with PureData through UDP. We present some applications, from the composition and interpretation of electro-acoustic scores to the interpretation of instrumental scores by technically limited musicians or impaired persons.

Keywords : *Computer Music*, *Temporal Relations*, *Constraints Programming*, *Interaction*

Table des matières

Première partie

Problématique et outils

Résumé

Nous exposons ici la question de l'interprétation en musique et expliquons en quoi celle-ci est très restreinte dans le cas de la musique électro-acoustique sur support. Nous dégageons une problématique autour des variations agogiques et analysons ces dernières pour savoir comment elles sont utilisées par les interprètes, et comment le compositeur en définit les limites à l'écriture de la pièce.

Nous proposons une analogie entre l'écriture des possibilités de variations agogiques et la définition de contraintes temporelles. Nous nous intéressons à l'utilisation de la programmation par contraintes en informatique musicale, et aux modèles temporels en composition assistée par ordinateur. De plus, nous présentons plus longuement un outil dont nous ferons usage dans la suite de ce mémoire : les réseaux de Petri. Comme nous utilisons ces réseaux pour représenter les relations temporelles des partitions et exécuter ces dernières, nous présentons à la fin de cette partie une structure particulière de réseaux concue pour représenter les ordres temporels : les "réseaux d'occurrences". Nous démontrons également des équivalences entre ce type de réseau et une algèbre temporelle, les S-langages.

Abstract

We present here what is the musical interpretation, and we explain why the electro-acoustic pieces recorded on a media cannot be interpreted. We arise a problem about the agogic variations, and we analyse it to discover how the musicians can use these variations and how the composer can write their framework.

We propose a comparaison between the way the composer defines the possibilities for agogic variations and the definition of a temporal constraints problem. We expose other uses of constraints programmation in computer music and some temporal models used in computer assisted composition. In addition, we insist on a specific tool that we use in this thesis : the Petri nets. Since we are interested in using these nets to represent the temporal relations between the events of the scores, and to execute the scores, we present at the end of this part a specific structure designed to represent the temporal order : the "occurrences nets". We also demonstrate some equivalences between these nets and a temporal algebra : the S-langages.

Chapitre 1

Contexte musical et problématique

1.1 Un historique de l'informatique musicale

L'objectif ici étant d'offrir une vision très générale de l'univers musical dans lequel le travail présenté se situe, nous ne survoleront que très rapidement l'histoire qui lie la musique et l'informatique. Pour un historique plus détaillée, on pourra se référer à [58].

1.1.1 Evolutions de la musique au XX^e siècle

En 1951, plus d'un demi-siècle après les premières découvertes scientifiques et techniques qui bouleversèrent radicalement nos conceptions du monde sonore et musical, Pierre Schaeffer[1] fonde à Paris le Groupe de Recherche de Musique Concrète qui deviendra par la suite le Groupe de Recherches Musicales[2]. Rejoint par d'autres chercheurs/compositeurs comme Pierre Henry et Pierre Boulez, cette institution sera le siège de toutes sortes d'expérimentations musicales s'appuyant sur les possibilités offertes par les avancées scientifiques et techniques de l'époque. On assistera finalement à une remise en question de notions musicales *a priori* évidentes comme le son, le timbre, l'écoute, etc.

La seconde moitié du XX^e siècle marque ainsi les débuts d'une alliance durable entre la composition musicale et la technique. Un nouveau langage musical, étroitement lié à l'évolution technologique, se développe peu à peu.

Ce lien provient de l'utilisation par les compositeurs de sons préalablement enregistrés, d'une matière sonore concrète. La composition se focalise alors sur le traitement de ces sons et sur leur organisation dans le temps. L'utilisation de procédés analogiques de l'époque, accorde une place centrale à la bande magnétique, à la fois outil de travail, et support de diffusion des pièces.

Les développements de l'informatique à partir des années 50, vont progressivement modifier les outils des compositeurs.

Dès 1957, Max Mathews réalise notamment la première synthèse de sons par ordinateur et malgré les temps de calcul importants des machines de l'époque, les capacités logiques et mathématiques de ces dernières laissent entrevoir un potentiel créatif considérable. La manipulation des sons sous forme de nombres conduit alors aux premiers programmes de montage et de traitement sonore, aux premiers outils informatiques de création sonore.

Il faudra attendre l'arrivée des mini-ordinateurs à la fin des années 70 pour que se développe un usage convaincant des logiciels en temps différé. Peu de temps après cette rupture technologique, l'apparition de processeurs câblés et du protocole MIDI[3] ouvre les portes de **la manipulation en temps réel**.

Dans les années 90, le musicien dispose désormais du micro-ordinateur et de nombreux logiciels d'une puissance sans cesse croissante, permettant ainsi l'intégration au sein d'une même machine de tous les

[1] (1910–1995) Ingénieur des télécommunications, compositeur, essayiste et écrivain, il crée le studio d'essai en 1942 et va donner naissance à la **musique concrète** (élaboration d'une pièce musicale issue d'un travail concret sur le matériau sonore).

[2] Connu aujourd'hui sous le nom INA-GRM (Groupe de Recherches Musicales de l'Institut National de l'Audiovisuel).

[3] Musical Instrument Digital Interface, standard de description numérique d'évènements musicaux simples.

outils souhaités : outils d'analyse, de Composition Assistée par Ordinateur, d'enregistrement, de montage, de mixage, de synthèse, de transformations sonores, de représentation graphique et enfin d'**interaction temps réel**.

Au cours de cette évolution des capacités des outils numériques, les pratiques de la composition ont peu à peu migré de l'analogique vers le numérique.

1.1.2 Musique Electro-acoustique

Parallélement à l'émergence de la musique concrète, un groupe de compositeurs autour de Karlheinz Stockhausen à Cologne, expérimentait l'utilisation systématique de la synthèse électronique pour donner naissance à la musique électronique. Les compositeurs de ce mouvement conçoivent alors la musique comme une activité purement intellectuelle et veulent maîtriser formellement tous les paramètres des sons sur la partition avant de les fabriquer.

L'apparition de sons de synthèse de plus en plus variés et la possibilité de transformer les sons *concrets* tout en conservant leur richesse expressive ont conduit à mêler les approches concrètes et électronique. C'est généralement sous la dénomination de *musique électro-acoustique* que l'on désigne ce rapprochement. Les pièces *électro-acoustiques* sont ainsi composées par organisation temporelle de sons[4] ; ces sons étant concrets (issus d'une captation) ou de synthèse pure, des traitements sur ce matériau de base étant très courants.

Comme dans le cadre de la musique concrète, une pièce *électro-acoustique* était à l'origine fortement liée à un support. Composer une pièce consistait à la fixer définitivement, "jouer" la pièce revenant à la diffuser.

La puissance des machines a toutefois permis d'intégrer des modifications en temps réel. Plus précisément, certaines pièces rendent possibles le contrôle de paramètres de synthèse ou de traitement pendant l'exécution.

Enfin, on peut également citer le cas de la musique dite *mixte*, mêlant parties *électro-acoustiques* et parties instrumentales.

Cette évolution de la musique a induit d'importantes modifications de sa notation et de son interprétation.

1.2 La notation

La notation de la musique permet de transcrire graphiquement les opérations nécessaires pour exécuter une œuvre.

Formellement, on peut citer Bennet [15] qui identifie trois rôles majeurs pour la partition :
- préciser ce que doit jouer l'interprète et à quel moment
- la préservation et la transmission des pièces
- l'analyse et la réflexion

Dans le cadre de la musique instrumentale, les deux premiers points sont possibles grâce à la préservation conjointe des instruments et des techniques qui s'y rattachent. Ceci assure la conservation en l'état des processus de production du son et du savoir faire lié au contrôle de leurs paramètres. Le maintien de ces connaissances permet le passage dans le domaine symbolique en représentant la musique comme une suite d'actions, décrites plus ou moins précisément, à effectuer sur des instruments. La figure 1.1 présente un exemple de partition instrumentale, il s'agit d'un extrait manuscrit de l'opéra *Les Troyens* composé par Hector Berlioz entre 1856 et 1858. Outre les symboles issus directement de la notation musicale, on trouve également des indications en français ("*Même mouvement, un peu animé*") qui viennent compléter la notation pour préciser l'intention du compositeur.

L'exemple donné sur la figure 1.1 correspond à un moment historique précis, et les symboles utilisés sont ceux d'un style et d'une époque déterminés. Dufourt [46] précise que la notation musicale s'est constamment redéfinie en fonction de nouveaux paramètres introduits dans la musique. Les profonds changements opérés aux XX^e siècle ont naturellement modifié en profondeur la notation de la musique. L'exploration de nouvelles possibilités avec les instruments a remis en cause les techniques classiques de

[4] Edgard Varèse préfère d'ailleurs parler de *sons organisés* plutôt que de musique concrète ou électro-acoustique.

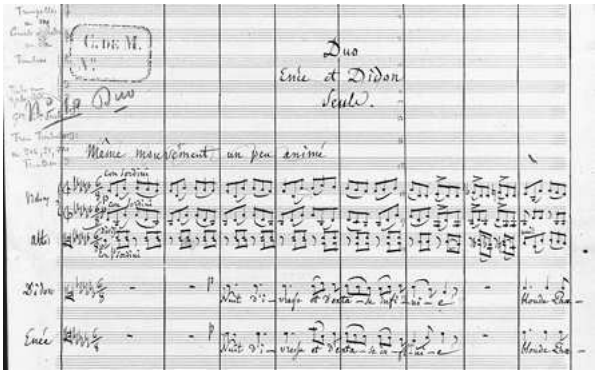


FIG. 1.1 – Un extrait d'une partition manuscrite des *Troyens* (1856-1858) d'Hector Berlioz

production du son. A défaut de proposer des nomenclatures pour chaque nouvelle manière de produire le son, dont certaines sont propres à une seule pièce, la notation s'est orientée vers des descriptions graphiques d'évolution des paramètres de production du son.

L'utilisation de processus de synthèse sonore électroniques, puis numériques, a encore accentué cette évolution de la notation vers la description de courbes de paramètres de synthèse. Pour une analyse des modifications de la notation musicale, inhérentes à l'introduction de la synthèse sonore dans les pièces, on pourra se référer à la thèse de Jean Bresson [22].

Dans le cadre de la musique électro-acoustique, si on exclut la musique mixte, les deux premiers points cités par Bennet n'ont plus besoin de la partition. Les pièces étant fixées sur support, il n'y a plus d'interprètes au sens de la musique instrumentale, et la transmission se fait par le biais du support. Le troisième point en revanche nécessite toujours une représentation de la musique. Sans entrer dans les détails, la visualisation d'une œuvre à des fins d'analyse par une tierce personne ou pour nourrir le processus de composition, est un rôle central de la notation musicale ; à ce sujet, on pourra se rapporter entre autre à [11].

A titre d'exemple, nous reproduisons sur la figure 1.2, un extrait d'une représentation de *Poèmes électroniques* d'Edgard Varèse, pièces intégralement sur support, composée de sons de synthèse et de sons concrets. Cette partition étant descriptive, les courbes présentent les variations de paramètres perceptifs (hauteur de son). Il est à signaler que dans cet extrait illustre un changement de notation du temps dans la mesure où l'évolution continue des paramêtres est explicitement représentée, à la différence de la notation classique. On notera au passage la présence de symboles issus de la notation classique pour indiquer des notions de nuances (f).

L'utilisation d'outils numériques et le développement d'interfaces graphiques pour ces derniers ont conduit dans certains cas à adopter des représentations liées à ces outils. Nous présenterons certains de ces outils dans la suite, cependant on peut donner ici l'exemple d'un séquenceur numérique, sur la figure 1.3. Il s'agit d'une capture d'écran du logiciel *Ardour*[5]. La représentation des sons se fait au travers de leur forme d'onde (l'amplitude en fonction du temps). Calqué sur les systèmes analogiques utilisant la bande magnétique (table de mixage, magnétophone ...) et inspiré de la notation classique par *portées*, celui-ci s'appuie sur une représentation linéaire du temps de gauche à droite et un regroupement des sons par *pistes* (structure s'étendant sur tout le morceau, présentant une séquence de son).

Sans pouvoir parler réellement de partition, un tel exemple constitue une représentation de la pièce dans le sens où on y trouve des courbes indiquant des évolutions de paramètres sonores au cours du temps.

La question de la représentation de la musique électro-acoustique reste encore un sujet ouvert, et de nouvelles représentations souvent liées à une œuvre ou à un outil particulier voient régulièrement le jour.

Une autre question, fortement liée à la notation musicale est celle de l'interprétation.

[5] http ://ardour.org/

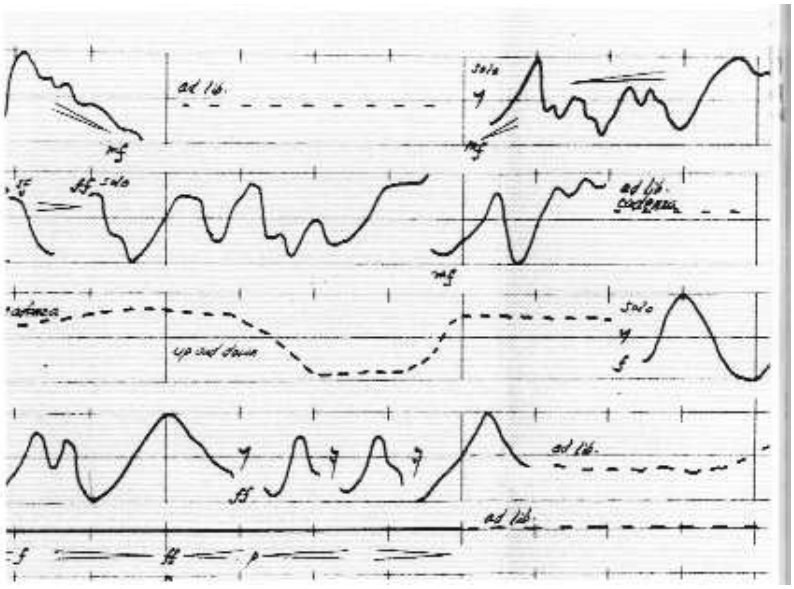


FIG. 1.2 – Un extrait d'une représentation de *Poèmes électriques* d'Edgard Varèse

FIG. 1.3 – Une capture d'écran d'une session du logiciel *Ardour*

1.3 Interprétation

L'interprétation d'une pièce est le processus par lequel un musicien fait vivre la notation musicale de la pièce. L'interprète d'une œuvre s'éloigne plus ou moins de la partition en modifiant divers paramètres de la pièce (rythme, hauteur de notes ...), afin de donner sa propre vision de la pièce. Certaines informations sont parfois absentes des partitions laissant l'interprète libre de les choisir. L'évolution de l'interprétation a ainsi été conjointe de celle de la notation musicale ([44]).

Cette évolution peut se résumer par un encadrement de plus en plus grand des libertés de l'interprète au fur et à mesure de l'élaboration de la notation musicale. Alors qu'à la Renaissance, il lui est possible de modifier fortement la notation, allant jusqu'à y ajouter des ornements, à partir du $XVIII^e$ siècle, il devient peu à peu contraint à une lecture précise de la partition.

Il faut toutefois noter que la partition constitue une approximation, ou la simplification de processus très complexes, et jouer exactement ce qui est écrit sur la partition est parfois tout simplement impossible. Ce cas de figure peut-être lié à des problèmes techniques liés à l'instrument. C'est par exemple le cas dans la *Chaconne pour Violon Seul*[6] de J.S Bach, dont le premier accord de quatre notes doit être arpégé (notes jouées séparément), car la configuration des cordes d'un violon font que seulement deux cordes peuvent être jouées simultanément. Ceci conduit les instrumentistes à utiliser les capacités harmoniques du violon pour simuler le jeu simultané des quatre notes.

D'autres impossibilités naissent des caractéristiques biomécaniques (liés à la morphologie) des instrumentistes. Dans la même chaconne de Bach, certaines notes ne peuvent être tenues aussi longuement que ce qui est écrit, car les changements de position nécessaires à l'exécution des notes suivantes étouffent le son. Un autre problème biomécanique classique est celui du *passage du pouce* au piano. Celui-ci consiste pour le pianiste à faire passer son pouce sous sa main, pour enchaîner une note jouée avec le pouce avec une note jouée avec un autre doigt (ou inversement). Or les études biomécaniques de McKenzie et VanEerd [66] ont montré que dans le cadre d'un jeu continu de notes de durée égales, ce type d'enchaînement perturbe la durée des notes. Ne pouvant se passer du pouce lors du jeu, le pianiste doit alors s'arranger avec la notation et jouer parfois des notes d'une durée différente de celle qui est écrite. La non-prise en compte de ces différentes spécificités fait qu'une partition est une approximation de la transcription exacte d'une réalisation instrumentale de l'œuvre.

Certains compositeurs ont, quant à eux, volontairement indiqué des éléments injouables dans leurs partitions. On trouve deux de ces exemples dans l'œuvre de Robert Schumann. Le premier se trouve à la fin des *Variations Abegg*[7], œuvre pour piano, dans laquelle le compositeur indique un accent au milieu d'une note tenue, alors qu'effectuer un tel accent est impossible au piano. Le second exemple, assez curieux, consiste en la notation sur la partition des *Humoreske*[8], œuvre pour piano, d'une ligne mélodique inaudible en plus des portées associées aux deux mains de l'interprète. L'exécution de cette ligne mélodique est inaccessible à tout pianiste normalement constitué, et de toute manière n'est pas destinée à être jouée. On trouvera des éclaircissements sur le sens de ces deux exemples et leur lien avec le mouvement *romantique* dans [87].

L'exécution fidèle de la notation n'est par conséquent pas possible et de toute manière pas souhaitable, le souffle de l'interprète donnant réellement vie à une œuvre. Ainsi, par choix et par nécessité, celui-ci prend des libertés avec la notation. Les aspects mécaniques et artistique de l'interprétation sont d'ailleurs souvent mélés, l'étude de Clarke *et al.* [27] au sujet des pianistes, a ainsi montré que le choix du doigté se faisait en premier lieu par rapport à l'interprétation. Des difficultés ou des impossibilités techniques se trouvent alors associées à une volonté artistique. Quelles que soient les motivations de l'interprète pour s'écarter de la notation, ces libertés restent définies et encadrées par la volonté du compositeur.

1.3.1 Possibilités et limites de l'interprétation

Les possibilités d'interprétation instrumentale d'une pièce dépendent de l'instrument avec lequel on la joue. Si des paramètres musicaux peuvent être modifiés indépendamment de l'instrument utilisé (rythme), d'autres, liés aux processus sonores, dépendent du degré de contrôle de l'instrument (vibrato). Dans [52],

[6] J.S. Bach *Chaconne extraite de la Partita n°2 pour violon(BWV 1004)*
[7] R. Schumann *Variations Abbeg - Op. 1* (1829-1830)
[8] R. Schumann *Humoreske - Op. 20* (1838)

Jean Haury analyse les possibilités d'interprétation avec des instruments à clavier. Il identifie quatre types de modification :

- *les variations dynamiques*, qui consistent à modifier le volume des notes sur toute une période.
- *l'accentuation*, qui sont des modifications locales du volume des notes.
- *l'articulation*, qui, métaphoriquement avec le langage parlé, est la manière dont le musicien va plus ou moins lier les notes entre elles.
- *les modifications agogiques*, qui sont les fluctuations passagères de rapidité ou de lenteur que l'on apporte au tempo. Il s'agit de décalages temporels du début ou de la fin de certaines notes par rapport au rythme écrit sur la partition.

Ces possibilités sont accessibles aux musiciens par l'intermédiaire de points d'interaction. Ces points sont des entrées dans la partition permettant de jouir des libertés offertes par l'interprétation. Dans le cadre des instruments à clavier, et d'un manière générale, pour les instruments à excitation non-entretenue, ces points de contrôle se situent sur les débuts et fins de notes. Pour les instruments à excitation entretenue, il est possible de modifier des paramètres sonores (et donc d'interpréter) au milieu d'une note.

Cependant, on peut noter que concernant les variations agogiques, les points d'interaction se situent toujours sur les débuts et fins de notes.

Chaque possibilité d'interprétation dispose de son type de point d'ineraction : un *accent* noté sur une note indique la possibilité de modifier le volume de cette note, un point d'orgue autorise l'interprète à modifier la durée de la note, ou encore l'indication *rubato* qui permet de s'écarter des durées de notes écrites pendant une certaine période.

Jean Haury précise également que la sémantique permet également de fixer les limites à ces libertés. Des indications de nuance par exemple (*pp*, *mf* ...) constituent des limites aux variations dynamiques, car elles restreignent le choix du volume de jeu. Dans le cadre des variations agogiques, l'indication d'un tempo limite l'éventail des vitesses d'exécution possibles.

Ainsi les possibilités d'interprétation d'une partition se traduisent par des libertés laissées à l'interprète, mais également par des limites. On voit d'ailleurs que l'évolution de la notation musicale a largement consisté à définir des signes pour exprimer ces limites, et ainsi contraindre de plus en plus l'interprète.

Les possibilités d'interprétation étant dépendantes des instruments considérés, on trouvera dans la thèse de Matthias Robine [86], d'autres exemples de paramètres sonores et musicaux modifiables lors d'une interprétation (timbre, vibrato ...), ainsi que des références vers des travaux les concernant.

1.3.2 L'interprétation de la musique électro-acoustique

On l'aura compris, l'interprétation musicale ne peut exister que dans la mesure où la notation ne prédétermine pas totalement la pièce, laissant une certaine part de sous-détermination sur le rendu sonore, incertitude qui sera levée par l'exécution du musicien.

Or dans le cadre des pièces électro-acoustiques sur support, cette part d'incertitude disparaît et les possibilités d'interprétation sont très restreintes. On peut citer la possibilité de spatialiser la pièce lors de sa diffusion en concert. L'interprète fait alors exister la pièce dans l'espace en répartissant le son sur les différents éléments du système d'écoute.

Certains compositeurs ont essayé de préserver dans leurs œuvres électro-acoustiques, cette incertitude qui est à la source de l'interprétation. On peut ainsi citer Philippe Manoury et sa formalisation des *partitions virtuelles* pour la synthèse sonore. Il s'agit ici de créer des pièces mettant en jeu des processus de synthèse sonore, dont certains paramètres sont indéterminés a priori, et tirent leur valeur de l'analyse en temps réel de la captation du jeu d'instrumentistes [67].

Une approche a consisté à construire des systèmes interactifs et à orienter ceux-ci vers l'écriture musicale. C'est notamment l'approche adoptée par Joel Chadabe, qui dans [26] développe l'idée d'*interactive composing*. L'interprétation est ici abordée comme une composition en temps réel, dont le rendu sonore est produit immédiatement par le système. L'utilisateur peut influer sur le déroulement de la pièce, ainsi que sur les paramètres sonores. En l'absence d'actions de l'utilisateur, le système génère automatiquement le contenu musical.

Certaines réflexions ont émergé concernant la possibilité de définir un système d'interprétation d'œuvres électro-acoustiques, comme par exemple celles de Kevin Dahan et Martin Laliberté [32]. Ici les auteurs cherchent à identifier les caractéristiques nécessaires à l'élaboration de pareil système. Plusieurs approches de l'interprétation sont évoquées, mais celle qui leur paraît la plus intéressante, est la modification en temps réel d'œuvres sur support. Les caractéristiques des pièces qu'ils imaginent pouvoir contrôler sont assez proches des paramètres modifiés lors d'une interprétation instrumentale (hauteurs, durées, dynamique, brillance, spatialisation). Cependant, il envisagent pouvoir appliquer ces modifications tout au long de la pièce, si bien que l'écriture de l'interprétation par le compositeur (ses possibilités et son cadre) n'apparaît qu'en filigrane. Pour les variations agogiques, le cadre est présenté comme implicite et les modifications peuvent intervenir à tout moment. La définition d'un système permettant la composition de pièces électro-acoustiques interprétables, dont le compositeur pourrait décrire les libertés laissées à l'interprète (leur étendue et leurs limites) n'a jamais été réellement abordée.

1.3.3 Problématique

Nous nous proposons dans cette étude de définir un système permettant l'écriture et l'exécution de pièces électro-acoustiques interprétables, considérées comme des ensembles de processus sonores temporellement organisés. Nous donnons à l'interprétation le sens qu'elle a dans le cadre de la musique instrumentale : des possibilités de modifier des paramètres de la pièce pendant son exécution, ces possibilités étant encadrées par la volonté du compositeur au travers d'une notation spécifique dans la partition de la pièce.

Nous nous limitons volontairement aux possibilités d'interprétation liées aux *variations agogiques*, et nous postulons que dans ce cadre, appliquer l'expression de l'intrprétation de la musique instrumentale à la musique électro-acoustique est possible. Ne nous intéressons qu'à des questions temporelles au niveau des notes, nous considérons les processus de production sonore des pièces, uniquement au travers de leurs propriétés temporelles. Nous ne détaillons donc pas les opérations réalisées pour produire le son.

Nous abordons cette problématique selon la méthode suivante. Nous commençons par analyser l'expression des libertés de variations agogiques et de leur limites dans les partitions de pièces instrumentales. Nous nous appuyons ensuite sur cette analyse pour proposer un formalisme de notation pour la composition de pièces électro-acoustiques interprétables. Nous cherchons ensuite à implémenter ce système pour être en mesure pas la suite de valider cette approche de l'interprétation auprès de compositeurs et d'interprètes

1.4 Formalisation des modifications agogiques

Nous cherchons ici à formaliser l'expression dans une partition instrumentale, des libertés d'interprétation liées aux modifications agogiques et celle de leur limites.

Ces modifications consistant en des variations temporelles sur les moments d'exécution des débuts et fins de notes, nous cherchons à déduire de la notation musicale de quelle manière une interprète peut modifier ces moments, et quelles propriétés ceux-ci doivent conserver lors d'une exécution.

Tout au long de cette formalisation, nous nous appuierons à titre d'exemple sur une mesure de la *Rapsody in Blue* de Georges Gershwin, dont une transcription est donnée sur la figure 1.4.

Nous donnons plus loin des diagrammes temporels d'exécution de cette mesure. Ceux-ci n'ont pas été obtenu par enregistrement et analyse d'exécutions par un interprète, mais imaginés à partir d'entretiens avec Jean Haury autour des techniques de l'interprétation.

1.4.1 Temporalité de la partition

Comme nous l'avons évoqué précédemment, une partition instrumentale traditionnelle présente une séquence d'opérations à réaliser sur un instrument, en vue de la production sonore. La temporalité y est codée au travers d'une flèche temporelle implicite, allant de gauche à droite, et dans laquelle viennent se positionner les éléments de la pièce. La position temporelle des notes dans la pièce s'exprime grâce à une subdivision du temps en unités logiques de taille fixe, appelés *temps* (pour éviter la confusion

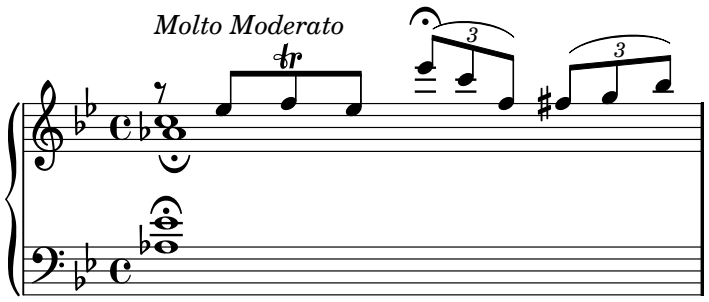


FIG. 1.4 – Un extrait de la *Rapsody in Blue* de Georges Gershwin

entre le temps au sens général et les *temps* au sens de pulsation musicale, nous metterons toujours ce dernier terme en italique). Cette subdivision s'appuie également sur des structures temporelles de plus haut niveau comme les *mesures* (séparées par des barres verticales). Une mesure représente un nombre déterminé de *temps* ; cette quantité de temps par mesure est transcrite par un chiffrage (dans l'exemple de la figure 1.4, le symbole C indique que la mesure dure 4 temps). Des structurations de plus haut niveau sont possibles. On peut voir dans la partition, une analogie avec un système calendaire, dans lequel des événements sont positionnés sur la flêche du temps grace à une unité de taille fixe.

Ainsi on peut, à lecture de l'extrait de la figure 1.4, affirmer que le premier fa commence au 2^{eme} temps de la mesure, et se termine sur le 2^{eme} temps et demi de cette même mesure. Dans ces conditions, il est possible de concéder à la partition, une vision figée du temps dans laquelle les dates des notes sont déterminées.

Pendant l'exécution de la partition, le musicien effectue une correspondancee entre le temps logique de la partition et le temps réel, afin de plonger l'œuvre dans le temps de l'écoute.

Par conséquent, selon l'approche statique de la partition, une interprétation consiste à lier la valeur des *temps* avec le temps de l'horloge (c'est à dire choisir un *tempo*), et à instancier les dates de début et de fin de chaque note proportionnellement à cette valeur de base.

Fort heureusement pour l'interprète, la conception calendaire du temps dans la partition peut voir varier la valeur de son unité de base. Ainsi, les dates logiques des débuts et fins de notes n'y sont pas exprimées dans l'absolu. En effet, la partition retranscrit des notes ordonnées et une durée pour chacune d'entre elle. Si bien que la date de début d'une note dépend de la durée des notes qui la précèdent, sa date de fin dépendant, elle, de sa date de début et de sa durée. La modification d'une date de début ou de fin d'une note, va ainsi entraîner le décalage des dates des notes qui la suivent. Si la partition donne une description de la pièce dans son ensemble avec l'organisation temporelle des notes, elle reste suffisamment laxiste sur certains aspects temporels pour ne pas totalement figer l'exécution par l'interprète. La correspondance entre temps logique et temps réel effectuée à l'exécution, est plus subtile qu'une simple proportion.

1.4.2 Formalisation mathématique

Partant du constat que l'interprétation consiste en une correspondance entre un temps logique de la partition et un temps réel de performance, on peut formaliser les possibilités et les limites de l'interprétation au travers d'égalités et d'inégalités impliquant les dates de débuts et de fins des notes.

Pour ce faire, nous introduisons la notion d'*événement*.

Définition 1.1 *Un* événement *d'une partition est soit le début soit la fin d'une note.*

Comme nous l'avons vu précédemment, les temporalités de la partition et d'une interprétation de celle-ci, s'expriment au travers des "dates" des événements. Dans un référentiel logique en ce qui concerne la partition, et dans un référentiel réel dans le cas de l'interprétation. Nous définissons alors deux fonctions correspondantes sur l'ensemble des événements d'une partition.

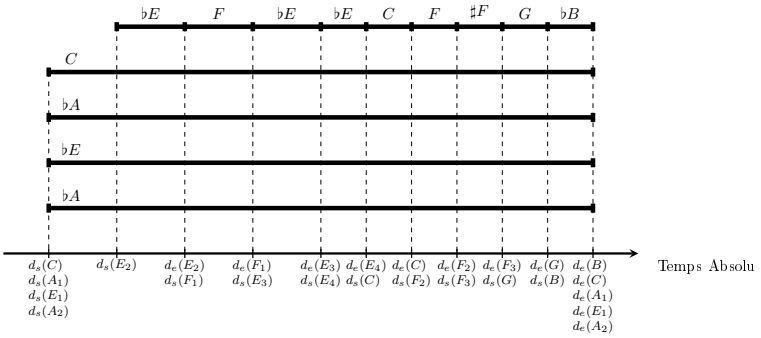


FIG. 1.5 – Un diagramme temporel d'une exécution de la mesure de la *Rapsody in Blue* de la figure 1.4. Sur cet exemple, il n'y a aucune interprétation, la date absolue de chaque événement est calculée proportionnellement à sa date logique selon une valeur de tempo, qui reste la même tout au long de la mesure.

Définition 1.2 *La fonction* lo-date *renvoie la date logique d'un événement, exprimée en* temps *depuis le début de la partition.*

Définition 1.3 *La fonction* re-date *renvoie la date réelle d'un événement au cours d'une interprétation de la partition, exprimée en secondes depuis le début de l'exécution.*

Une interprétation se caractérise donc par l'ensemble des dates réelles des événements de la partition. Nous l'avons évoqué plus haut, en absence d'interprétation, les dates réelles des événements sont directement proportionnelles à leurs dates logiques. Sachant que la valeur du tempo indique le nombre de *temps* par minutes, dans ce cas pour tout événement e, on a :

$$re - date(e) = \frac{60}{t}.lo - date(e)$$

La figure 1.5 présente un diagramme temporel, représentant les dates réelles des événements dans le cas d'une exécution sans interprétation de la mesure de la figure 1.4. Les notes sont ici identifiées selon la notation américaine et leurs événements sont représentés par les fonction *start* et *end*. Les dates réelles sont ici proportionnelles aux dates logiques selon un tempo supposé constant pendant toute la mesure.

Une telle exécution n'est guère réaliste et dans le cas d'une interprétation, les valeurs des dates réelles s'éloignent plus ou moins des ces valeurs parménidiennes. Comme dans le cadre des modifications agogiques, les points de contrôle se situent sur les événements de la partition, les possibilités d'interprétation consistent donc à choisir une valeur de date réelle pour chacun des événements.

Le cadre imposé à l'interprète se définit alors sous forme de relations qualitatives qui traduisent un ordre entre les événements, et de relations quantitatives qui limitent les valeurs des intervalles temporels entre les intervalles.

Relations qualitatives

Ces relations traduisent l'ordre des événements dans la partition.

La figure 1.6 propose le diagramme temporel d'une interprétation de la mesure de la *Rapsody in Blue* de la figure 1.4. Sur cet exemple, nous supposons que l'interprète joue *staccato* a première partie de la mesure, c'est à dire qu'il détache les notes. A l'inverse, la seconde partie de la mesure est interprétée *legato*, ce qui signifie que les notes sont liées, et celles-ci se chevauchent pendant l'exécution. De plus, l'interprète utilise le point d'orgue pour retarder la fin du *mi bémol*. Dans la réalité, les durées de chevauchement des notes pour le *staccato* et de détachement pour le *legato* dépendent de "l'école" à laquelle se rattache l'interprète. Les maîtres à l'origine de ces écoles ont ainsi théorisé plus ou moins précisément ces durées. Dans nos diagrammes, nous n'avons pas cherché à respecter les enseignements d'une école en particulier.

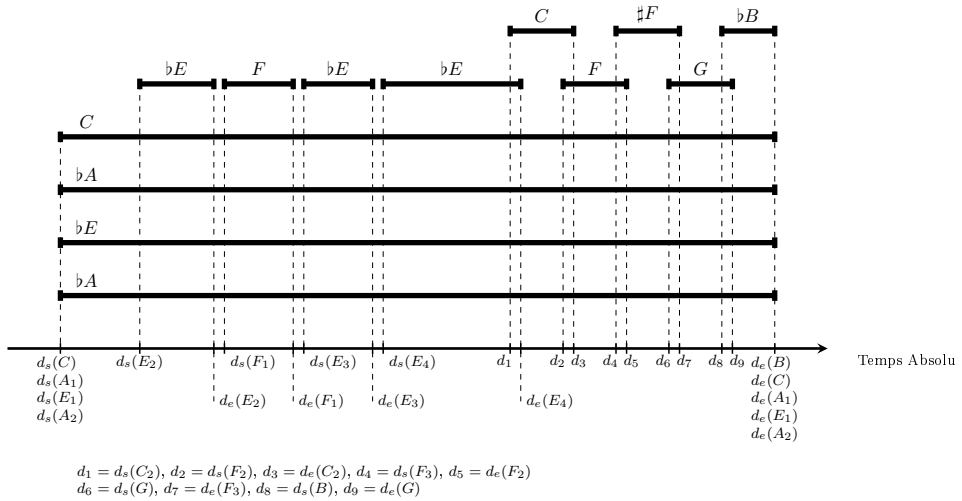


FIG. 1.6 – Le diagramme temporel d'une interprétation de la mesure de *Rapsody in Blue* dans laquelle le musicien joue *staccato* la première partie de la mesure. Les notes de cette partie sont donc séparées. De plus, il utilise le point d'orgue sur la fin du mi bémol pour prolonger cette note. Enfin, la seconde partie est jouée *legato*, les notes de cette partie se chevauchent.

Le premier constat que l'on peut faire est que certaines égalités valables dans le temps logique de la partition ne sont plus vérifiées dans le temps réel de l'interprétation. En particulier, pour deux notes n_1 et n_2 :

$$lo-date(end(n_1)) = lo-date(start(n_2)) \not\Rightarrow re-date(end(n_1)) = re-date(start(n_2))$$

La relation entre les dates réelles de ces événements dépend en fait du choix d'*articulation*, ainsi :
– pour le *staccato* :

$$re-date(end(n_1)) < re-date(start(n_2))$$

– pour le *legato* :

$$re-date(start(n_2)) < re-date(end(n_1))$$

Par conséquent, en l'absence d'indication concernant l'articulation comme c'est le cas sur l'exemple, aucune relation n'est fixée entre la fin d'une note et le début de la note qui la suit. A l'inverse, si indication d'articulation il y a, les inégalités ci dessus sont imposées.

En outre, il est possible d'identifier des relations qui se vérifient pour chaque exécution.

Tout d'abord il existe une relation que nous qualifierons de *relation de cohérence*.

Propriété 1.1 *Pour toute note n :*

$$re-date(start(n)) \leq re-date(end(n))$$

De plus, si deux notes se suivent on peut déduire une relation sur leurs dates de début :

Propriété 1.2 *Soient n_1 et n_2, deux notes :*

$$lo-date(end(n_1)) = lo-date(start(n_2)) \Rightarrow re-date(start(n_1)) \leq re-date(start(n_2))$$

Enfin, on peut également remarquer une autre relation concernant les dates de débuts de notes.

Propriété 1.3 *Soient n_1 et n_2, deux notes :*

$$lo-date(start(n_1)) = lo-date(start(n_2)) \Rightarrow re-date(start(n_1)) = re-date(start(n_2))$$

Il nous faut cependant citer une exception à cette dernière relation : le *leed*. Il s'agit d'un type d'interprétation particulier utilisé par les pianistes pour les pièces romantiques, celui-ci consiste à désynchroniser les deux mains à certains moments et à introduire un délais de quelques millisecondes entre la mélodie et la basse [78].

Sans chercher à faire une liste exhaustive de toutes les relations qu'il est possible d'inférer à partir d'une partition, on peut montrer que le cadre de l'interprétation s'exprime au travers de ces relations. Ainsi dans l'exemple d'interprétation de la figure 1.6, si le musicien utilise le point d'orgue pour retarder la fin du *mi bémol*, ce retard va se propager au reste de la mesure, et décaler les dates des notes suivantes. Cette propagation étant due au respect des relations entre les événements. Car prendre en compte la modification introduite par le point d'orgue tout en respectant les relations, conduit à modifier les dates des notes suivant le point d'orgue.

Intervalles de temps

Le cadre de l'interprétation s'exprime également au travers des intervalles de temps séparant les événements d'une partition. En formalisant cette notion d'intervalle, on peut définir deux fonctions donnant la valeur d'un intervalle dans le temps logique de la partition et dans le temps réel de l'interprétation.

Définition 1.4 *Soit I un intervalle séparant deux événements e_1 et e_2,*

$$lo-val(I) = lo-date(e_2) - lo-date(e_1)$$

$$re-date(I) = re-date(e_2) - re-date(e_1)$$

Dans ces conditions, la relation entre les dates logiques et les dates réelles au travers du tempo peut s'exprimer ainsi.

Propriété 1.4 *Lors de l'exécution de la partition à un tempo t, en l'absence d'interprétation, pour tout intervalle I :*

$$re-date(I) = \frac{60}{t}.lo-date(I)$$

La modification du tempo au cours de l'exécution, va naturellement conduire à modifier les valeurs réelles des intervalles comme sur l'exemple de la figure 1.7. Sur cet exemple, les choix d'articulation sont les mêmes que sur l'exemple de la figure 1.6, *staccato* au début, *legato* à la fin, avec retard de la fin du *mi bémol* grâce au point d'orgue. Cependant, le tempo est accéléré pendant la première partie de la mesure, tandis qu'il est ralenti pendant la seconde.

Cependant, les valeurs possibles pour le tempo sont limitées par l'indication d'humeur en début de partition : *molto moderato*. Les musiciens et chefs d'archestre estime que cette indication d'humeur, doit se traduire par une valeur de tempo comprise entre 80 et 100 battements par minute.

Cela conduit donc à imposer des valeurs pour les intervalles de la partition. En prenant par exemple, l'intervalle I séparant les débuts des deux premières notes (bE et F), alors :

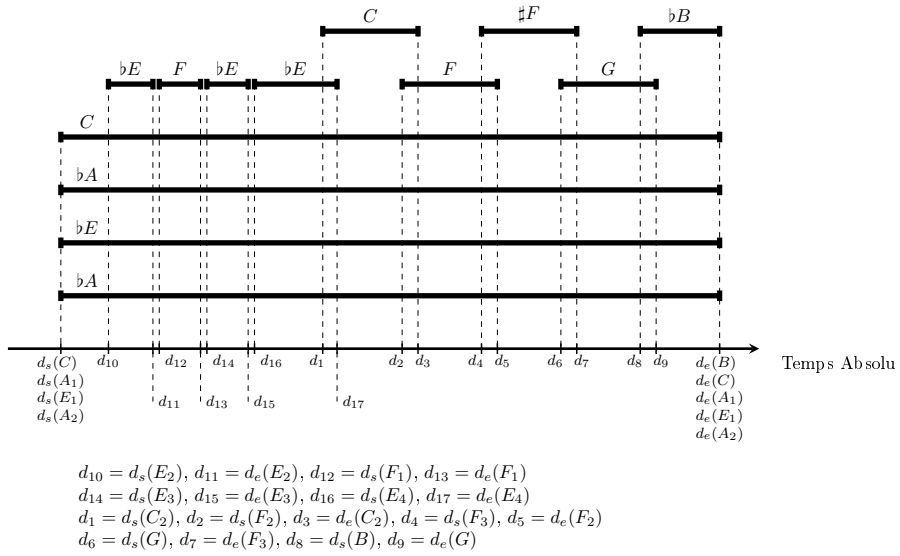
$$lo-val(I) = 1$$

et

$$\frac{60}{100} \leq re-val(I) \leq \frac{60}{80}$$

Si l'indication de tempo restreint les possibilités, le point d'orgue, lui, ouvre largement le champ des possibles. Ainsi si une note n dispose d'un point d'orgue, l'intervalle représentant la durée de n n'a pas de limites à priori :

$$0 \leq re-date(end(n)) - re-date(start(n)) \leq \infty$$



$d_{10} = d_s(E_2)$, $d_{11} = d_e(E_2)$, $d_{12} = d_s(F_1)$, $d_{13} = d_e(F_1)$
$d_{14} = d_s(E_3)$, $d_{15} = d_e(E_3)$, $d_{16} = d_s(E_4)$, $d_{17} = d_e(E_4)$
$d_1 = d_s(C_2)$, $d_2 = d_s(F_2)$, $d_3 = d_e(C_2)$, $d_4 = d_s(F_3)$, $d_5 = d_e(F_2)$
$d_6 = d_s(G)$, $d_7 = d_e(F_3)$, $d_8 = d_s(B)$, $d_9 = d_e(G)$

FIG. 1.7 – Le diagramme temporel d'une interprétation de la mesure de *Rapsody in Blue*, dont les choix d'interprétation sont les mêmes que pour l'exemple ci-dessus (*staccato* pour la première partie, *legato* pour la seconde), la différence est que le tempo est modifié. La première partie de la mesure est jouée plus rapidement, tandis que la seconde est jouée plus lentement.

Le point important à retenir ici est que le compositeur peut fixer les limites de l'interprétation en imposant des relations qualitatives et quantitatives sur les dates des événements de la partition. Ces relations peuvent être perçues comme de véritables contraintes sur les valeurs des dates réelles des événements. Un interprétation de la pièce se devra alors nécessairement de respecter ces contraintes. Ce constat nous conduit donc à envisager un formalisme de représentation de partitions, dans lequel le cadre de l'interprétation est fixé grâce à des contraintes qualitatives et quantitatives sur les dates des événements.

Chapitre 2

Temps et contraintes en informatique musicale

Nous présentons ici les approches temporelles de certains outils de l'informatique musicale. Si tous ne sont pas précisément destinés à la Composition Assistée par Ordinateur en tant que telle, nous cherchons à exposer un éventail de conception du temps en informatique musical. Nous exposons ensuite des travaux utilisant des systèmes de contraintes en informatique musicale. Notre objectif est ici d'identifier quelles approches de représentation musicale pourraient s'avérer pertinentes pour le système que nous cherchons à édifier.

2.1 Formalismes pour la composition

Loin de faire une liste exhaustive des logiciels développés dans le cadre de l'informatique musicale, nous donnons ici quelques exemples représentatifs.

2.1.1 Séquenceurs

Dans le chapitre 1, nous avons rapidement évoqué le cas des séquenceurs. Si ceux-ci ne sont pas à proprement parlé des outils de CAO, ils constituent un élément important dans la création musicale, en particulier pour la musique électro-acoustique. Sur la figure 1.3, nous présentons la capture d'écran d'une session du logiciel *Ardour*. Ce logiciel libre est tout à fait emblématique de l'ensemble des séquenceurs. Ceux-ci sont issus de la numérisation du matériel des studios d'enregistrement analogiques. La bande magnétique est remplacée par l'espace disque de l'ordinateur, tandis que l'interface graphique du logiciel, reproduit assez fidèlement la configuration et les fonctionnalités d'une table de mixage. Seule différence de taille avec l'analogique, les sons sont représentés dans le temps, le long d'une ligne de temps horizontale. Cette représentation conserve l'organisation par pistes de la table de mixage. La plus part d'entre eux permettent de mêler des sons (représentés par leur forme d'onde) avec des notes symboliques MIDI. La temporalité est ici figée comme pour les pièces sur support.

Exception notable dans le monde des séquenceurs, le logiciel *Live*[9]. Pour moitié, celui-ci se présente comme un séquenceur classique permettant l'édition et la représentation au travers de pistes et d'une ligne de temps linéaire. Cependant, dédié à la performance en concert (d'où son nom), *Live* intègre également une partie permettant la définition de boucles dans une temporalité événementielle et cyclique. La figure 2.1 présente une capture d'écran de cette partie.

On retrouve une organisation par pistes, non plus horizontales, mais verticales. Sur une piste (colonne), chaque ligne représente un bloc élémentaire qu'il est possible d'exécuter une fois avant de passer au suivant (la flèche temporelle va de haut en bas), ou qu'on peut exécuter en boucle, jusqu'au passage à la suite contrôlé par l'utilisateur. En outre il est possible de construire une boucle à partir de plusieurs blocs

[9] http ://www.ableton.com/

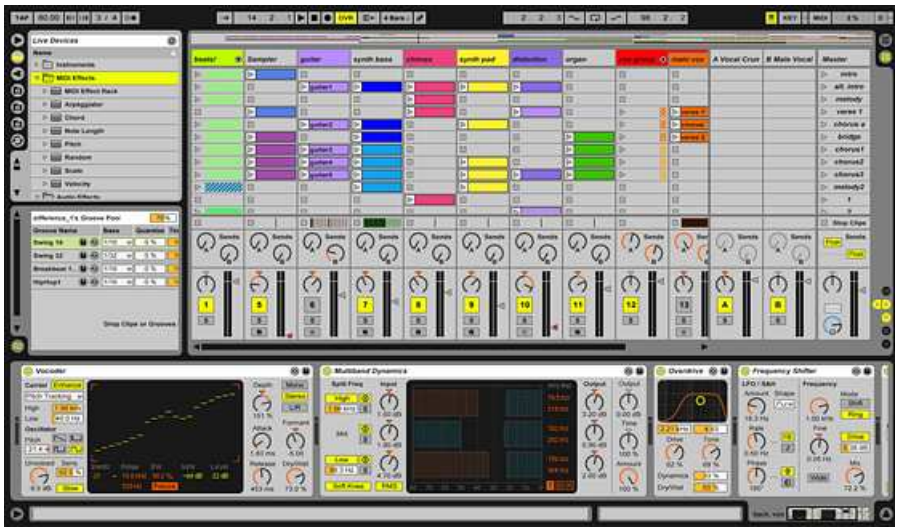


FIG. 2.1 – Une capture de l'éditeur de boucles de *Live*

successifs et de définir des relations de synchronisation/attente entre les blocs de pistes différentes. Cela conduit pendant l'exécution à des déroulements temporels non linéaires. L'originalité du logiciel est de pouvoir exécuter simultanément des parties écrites dans les deux référentiels temporels, l'ensemble des boucles se comporte alors comme un système interactif, dont on peut déclencher les événements par dessus le déroulement de la partie écrite dans le séquenceur "classique".

2.1.2 Langages et Environnements

Une tendance forte de l'informatique musicale et assez appréciée des compositeurs et autres utilisateurs, est la conception de langages de programmation dédiés. L'intérêt de ce type d'approche est de ne pas trop spécialiser les logiciels en laissant au langage un vaste pouvoir d'expression, tout en le maintenant accessible à des non-informaticiens. Ceci pour lui permettre de satisfaire des utilisateurs aux différentes préoccupations et manières de voir.

Langages pour la synthèse sonore

Depuis l'historique famille des *Music N* de Max Mathews [69] ayant donné entre autre le célèbre *CSound*, la synthèse sonore numérique a proposé nombre de langages de programmation, et on en trouve pour les différents procédés de synthèse. Nous nous arrêtons ici sur deux exemples bénéficiant du développement de la puissance de calcul pour proposer des processus de synthèse en temps réel, contrôlable par l'utilisateur.

Créé par Miller Puckette, le système *Max/MSP*[10] [81] est un langage de programmation visuel permettant d'écrire des programmes de synthèse et traitement sonore. Un programme est appelé *Patch*, un exemple en est donné sur la figure 2.2.

Un *Patch* représente l'arbre d'appels du programme (relations de causalité), l'exécution d'un programme étant déclenché par une excitation A l'origine, ces excitations provenaient de flux d'événements MIDI, puis le système a été étendu pour permettre l'excitation par des flux de signal audio. On peut également agir directement sur l'interface graphique pour déclencher une exécution, ou envoyer au système des messages réseaux, plus nécessairement MIDI. *Max/MSP* adopte donc une conception temporelle totalement événementielle, et on retrouve dans la gestion de flux propre au système, la métaphore du fleuve d'Héraclite. Si *Max/MSP* est très puissant et très apprécié pour la conception de systèmes interactifs,

[10] http ://www.cycling74.com/

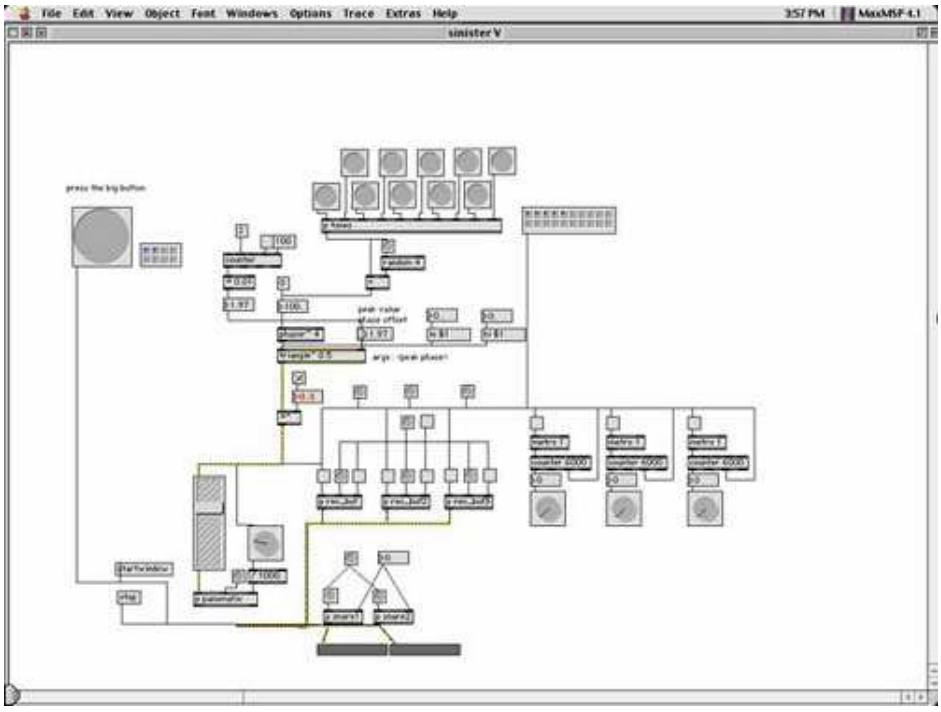


FIG. 2.2 – Un exemple de *Patch* dans *Max/MSP*

l'absence de possibilités réelles d'écriture du temps, reste une de ses limitations fortes pour une utilisation compositionnelle.

Son concepteur à chercher par la suite à pallier ce problème au travers d'un autre système, *PureData*[11] [82], dont un des objectifs est justement d'apporter la possibilité d'écrire le temps au travers de structures temporelles appelées *data structures*. *Pure Data* reprend donc le paradigme de programmation visuelle pour la gestion de flux de *Max/MSP*, en y ajoutant une interface d'écriture basée sur une ligne de temps [83]. La figure 2.3 présente une capture de cette interface.

Il s'agit au travers de formes graphiques, de représenter l'ordonnancement et la mise en temps de flux de paramètres qui, au cours de l'exécution, seront communiqués à la partie calcul du système. L'utilisation des processus de synthèse est ainsi inscrite temporellement. Comme *Live* partait d'une représentation linéaire pour y ajouter une conception événementielle, *PureData* part à l'inverse d'une approche purement événementielle pour y introduire une écriture linéaire du temps, et ce afin d'obtenir cette hybridation propre à l'écriture de pièces modifiables à l'exécution.

Composition Assistée par Ordinateur

Les systèmes purement dédiés à la CAO ont naturellement dû développer des approches de la temporalité qui leur sont propres. Si les langages de programmation dédiés à la CAO ont abordé le temps d'un point de vue figé, on trouve quelques approches intéressantes dans les langages et environnements de programmation visuelle pour la CAO. Ce type de langages a connu un développement important au sein de l'Ircam à partir des années 80.

Nous nous attardons ici sur l'environnement de composition *PatchWork* [65] et un de ses successeurs : *OpenMusic* [2], [12] tout deux basé sur le langage *Lisp*. Comme dans *Max/MSP*, les programmes créés dans *PatchWork* et *OpenMusic* sont appelés des *patchs*, et leur représentation graphique présente un graphe d'appel entre les fonctions mises en jeu dans le programme. Un exemple de *patch* est présenté sur la figure 2.4.

[11] http ://puredata.info/

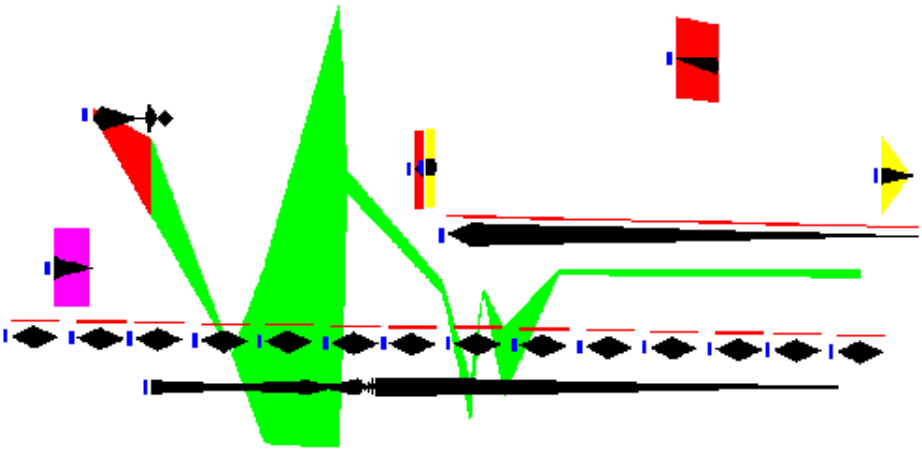


FIG. 2.3 – Un exemple d'utilisation des *data structures* de *Pure Data*

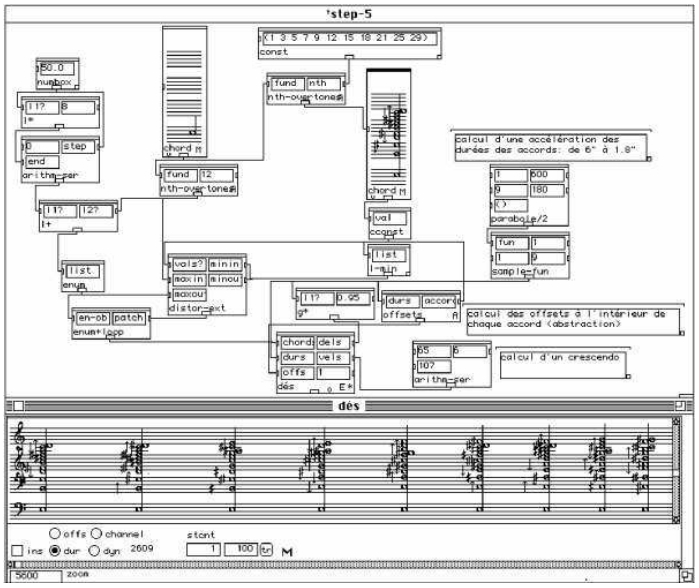


FIG. 2.4 – Un exemple de patch dans *PatchWork*

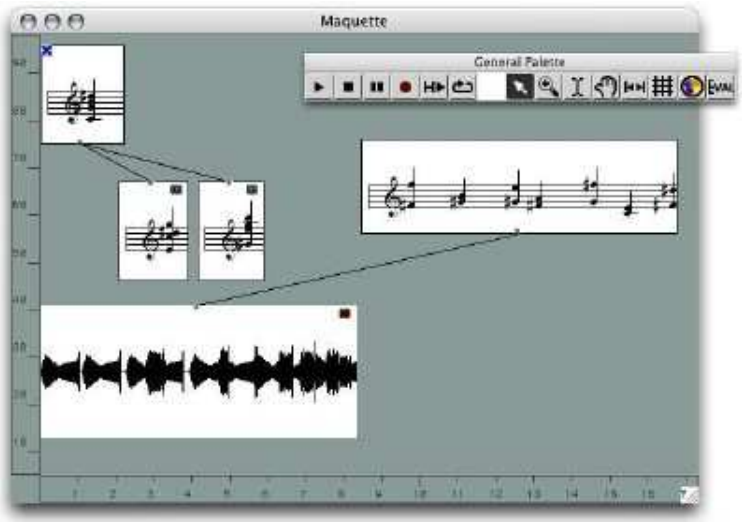


FIG. 2.5 – Un exemple de *Maquette* dans *OpenMusic*

Cependant le paradigme sous-jacent est très différent de celui de *Max/MSP*, ici les programmes sont des expressions en langage *Lisp* qui sont évaluées pour produire des résultats symboliques (partition instrumentale, notes midi ...) ou des valeurs. Les patchs permettant ainsi d'effectuer des calculs en temps différé pour créer un matériau symbolique ou sonore. L'évolution de *PatchWork* vers *OpenMusic* a été motivée par le développement de la programmation par objets qui est un paradigme central dans cet environnement qui s'appuie sur la couche *CLOS*[12] de *Common Lisp*. On trouve dans *OpenMusic*, un environnement, appelé les *Maquettes*, permettant de disposer dans le temps le matériau sonore calculé. Dans cet environnement, les objet résultant du calcul effectués par les *patchs* sont disposés sur une ligne temporelle horizontale, de gauche à droite. La figure 2.5 présente un exemple de *maquette* dans *OpenMusic*.

Il est important de noter qu'une *maquette* est elle même le résultat d'un calcul. Ce calcul peut être effectué lorsque l'utilisateur édite la *maquette* (modifie la place et la durée des objets depuis l'interface de la *maquette*), ou depuis un patch dont le calcul produit la *maquette*. Le temps devient alors un paramètre du calcul. On peut également noter qu'il est possible de définir des relations temporelles simples, par l'intermédiaire de marqueur fixé sur la ligne de temps, que l'on peut associer à des débuts ou fins d'objets.

Plus récemment, a été développé dans *OpenMusic* un système permettant la manipulation de représentation temporelles hétérogènes (symbolique et réelle) au travers d'une nouvelle interface : le *sheet*.

Citons enfin, une approche originale, celle du système *Iannix* [29], inspiré des travaux du compositeur Iannis Xénakis. L'idée est ici de proposer une représentation non plus au travers d'une ligne, mais d'un espace. Les axes d'un espace à deux dimension ont une signification temporelle. Nous présentons sur la figure 2.6, une capture d'écran du système.

Dans l'espace temporel, sont disposés des objets , ceux-ci peuvent représenter des courbes de valeurs ou bien des déclenchements. Une fois ces objets placés, l'utilisateur peut ajouter des lignes de temps sous forme de segments ou de courbes fermées. Ces lignes vont être parcourues par des curseurs, disposant chacun d'une vitesse propre, lorsqu'un curseur rencontre un objet, la valeur de courbe ou le déclenchement correspondant au point de rencontre entre le curseur est émis. Des raffinements sophistiqués sont possibles quant aux propriétés des curseurs : taille des curseurs, sens de parcours des lignes de temps, comportement en cas de rencontre des curseurs ...). *Iannix* émet des messages réseaux associés aux événements nés de la rencontre d'un curseur et d'une courbe, il permet ainsi de contrôler des processus de synthèse (au travers de *Max/MSP* par exemple). Ceci fait de *Iannix*, un système entièrement lié à l'écriture du temps, adoptant une approche tout à fait originale.

[12] Common Lisp Objects System

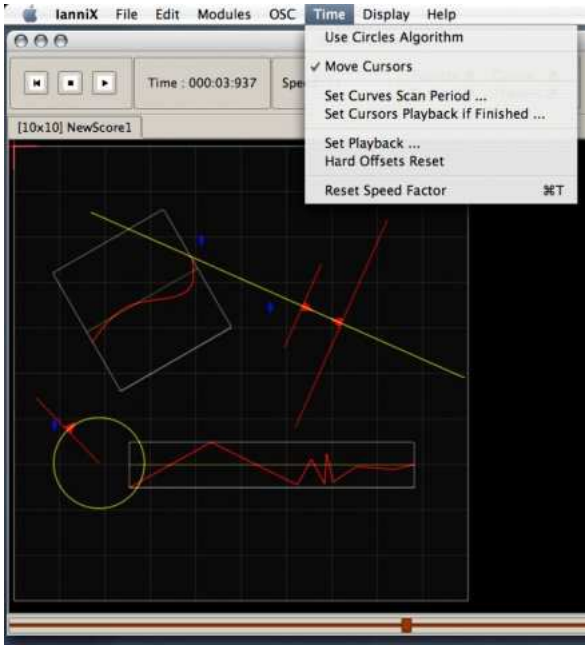


FIG. 2.6 – Une capture d'écran de *Iannix*

2.2 Musique et Contraintes

2.2.1 Programmation par contraintes

La programmation par contraintes est un cas particulier de programmation déclarative, dans laquelle le problème est exposé sous forme de *CSP* (Constraints Satisfaction Problem). Un CSP est constitué d'un ensemble de variables et de relations logiques entre ces variables. Résoudre le problème revient à trouver une valeur pour chaque variable, telle que l'ensemble de relations logiques sont satisfaites par ces valeurs

Formellement, un CSP se définit par un ensemble de variables $\{V_1, \ldots, V_n\}$ dont les valeurs appartiennent à des domaines $\{D_1, \ldots, D_n\}$, qui peuvent être finis ou infinis, et d'un ensemble de contraintes, que l'on peut définir comme des fonctions booléennes sur des sous-ensembles de variables, c'est à dire :

$$\otimes_{i \in I} D_i \rightarrow \{0, 1\}$$

avec I, un sous ensemble de $1, \ldots, n$.

La programmation par contraintes propose différentes méthodes permettant de résoudre des CSP. Une méthode est souvent adaptée à un type de problème particulier. Il n'existe par de méthode générique efficace pour résoudre un CSP. Une technique consiste ainsi à diviser le CSP en sous-problèmes caractéristiques, de résoudre chacun des ces sous-problèmes par une méthode spécifique et collecter ces résultats pour construire une solution globale.

La programmation par contraintes s'est vue appliquée dans de nombreux domaines. Depuis les interfaces graphiques où elle a beaucoup été utilisée jusqu'aux problèmes d'ordonancement, en passant par l'informatique musicale. Le lecteur souhaitant approfondir la question pourra se rapporter à [62] ou [68].

2.2.2 Contraintes dans l'informatique musicale

L'informatique musicale a trouvé dans la programmation par contraintes un formalisme à même de répondre à certains de ses besoins. En effet, la théorie musicale est constituée de nombreuses règles assimilables à des contraintes. Par conséquent des problèmes classiques liés à la composition sont formalisables par l'analyse et la résolution de systèmes contraints. De plus les applications historiques de la programmation par contraintes dans le domaine des interfaces utilisateurs ont suscité l'intérêt des développeurs

de logiciels de CAO. Nous ne prétendons pas ici faire un historique complet de l'utilisation des contraintes en informatique musicale, mais non présentons quelques travaux notables.

Différents solveurs

L'exemple historique, et qui a suscité le plus de développements, est celui de l'harmonisation automatique. Les règles de l'harmonie font en effet partie des éléments de théorie musicale modélisable sous formes de contraintes. A la suite de Kemial Ebcioglu qui dans son système *CHORAL* utilisait la programmation par règles pour créer des chorales à 4 voix dans le style de J.S. Bach, de nombreux travaux ont utilisé la programmation par contraintes pour résoudre ce type de problèmes. Citons les travaux de Pierre Roy qui avec son système *BackTalk* [88] propose l'harmonisation de basses chiffrées et de mélodies. On trouvera une présentation complète de l'utilisation de contraintes pour l'harmonisation dans [77]

L'attrait des compositeurs pour la programmation par contraintes a conduit à intégrer des solveurs permettant la résolution de problèmes de moins en moins spécifiques. Michael Laurson a été intégré à *Patchwork* un solveur de contraintes nommé *PWConstraints* [64]. Celui-ci permet de spécifier un problème par rapport aux propriétés que doivent vérifier une solution. Rueda et Bonnet ont quant à eux écrit un solveur, *Situation*[17], initialement conçu pour *Patchwork* puis intégré à *OpenMusic*. Il permet, au travers de fonctions du langage visuel d'*OpenMusic*, de résoudre des problèmes harmoniques et rythmiques.

Les travaux de Charlotte Truchet [92], et le solveur *OMClouds* qui en découlent, permettent de disposer dans *OpenMusic* d'un solveur plus général pour résoudre d'autres problèmes que les classiques questions d'harmonie et de rythme. Ce travail est notable à plusieurs titres. Tout d'abord, pour coller à l'environnement visuel d'*OpenMusic*, les contraintes sont représentées sous forme de *Patches*. L'utilisateur dispose alors d'une représentation visuelle des contraintes, ainsi que du graphe de contraintes associés à un problème. En outre, en plus d'un solveur classique, *OMClouds* s'appuie sur un second solveur basé sur la technique de recherche locale appelée *recherche adaptative* proposée par Philippe Codognet [28]. Cette heuristique permet de donner des solutions approchées à un problème de contraintes. Ceci a plusieurs avantages : proposer une solution à un problème ne disposant pas de solution exacte, et contrôler le temps calcul de l'exécution de l'algorithme ; la résolution peut-être interrompue à un moment donné tout en fournissant un résultat. L'utilisation de cette méthode a permis de considérer des problèmes d'analyse musicale et non plus uniquement des questions liées à la composition.

Spatialisation par propagation de contraintes

Une autre approche particulière est adoptée par Olivier Delerue [33], pour l'élaboration d'un système de spatialisation. Il s'agit ici de décrire la spatialisation d'un ensemble de pistes, au travers de la disposition virtuelles d'instruments produisant le son de ces pistes. La figure 2.7 donne un exemple de capture d'écran de *MusicSpace*. Le logiciel contrôle un système d'écoute afin de reproduire l'effet de spatialisation qui résulterait au oreilles de l'auditeur, représenté par l'avatar central, du jeu des instruments placés au niveau de leurs avatars.

L'originalité de ce système réside dans la possibilité de définir entre les instruments, des contraintes spatiales (distance l'un par rapport à l'autre, angles par rapport à l'auditeur ...). De plus, l'utilisateur peut en temps réel modifier la position d'un instrument, le système se chargeant de modifier la position des autres instruments en conséquence, dans le but de maintenir définies au préalable. D'un point de vue technique, le système utilise un algorithme de propagation de contraintes, qui permet d'une part de construire une solution pour obtenir la situation de spatialisation initiale respectant les contraintes. Puis de propager les modifications effectuées par l'utilisateur qui viennent perturber la solution initiale. Cette propagation aboutissant à une nouvelle solution du système de contraintes.

Contraintes temporelles

Outre les problème rythmiques, l'utilisation de contraintes pour résoudre des problèmes temporels liés aux dates d'événements musicaux a été tout particulièrement développé au sein du projet *Boxes* [16]. *Boxes* se présente comme un environnement de composition assistée par ordinateur, d'une représentation visuelle proche des *Maquettes* d'*OpenMusic* : les sons composant la pièces sont représentés sous forme

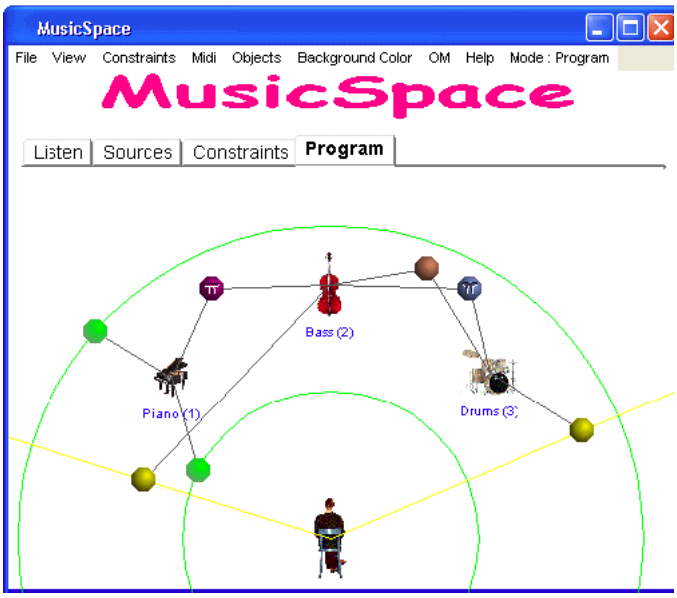


FIG. 2.7 – Une capture d'écran du logiciel *MusicSpace*

de boîtes, placés sur une feuille blanche dont l'axe horizontal représente le temps orienté (de gauche à droite). La figure 2.8 présente une capture d'écran du logiciel.

L'originalité provient du fait que le compositeur a la possibilité de définir entre les sons des contraintes temporelles exprimées à l'aides des relations de Allen [4]. Dans cette approche, on considère que le plus important pendant la phase de composition n'est pas la place absolue d'un son dans le temps, mais sa place relative aux autres sons. Ainsi lorsque le compositeur décide de modifier une boite (sa date de début ou sa durée), le système se charge de modifier les autres boîtes afin de prendre en compte la modification du compositeur et de respecter les relations temporelles définies. Ces relations sont donc à la fois l'expression d'une volonté musicale du compositeur, mais également une aide, lui permettant d'affiner le placement des son dans le temps en étant assurer leurs positions relatives seront assurées. *Boxes* utilise une bibliothèque de résolution de contraintes d'égalités et d'inégalités linéaires basée sur une résolution de simplexe (Cassowary [13]). Une fois la composition terminée, le morceaux est figé. Concernant la synthèse des sons, *Boxes* utilise la méthode SAS [36] qui autorise des changements dans la durée des sons.

Aujourd'hui

Plus récemment, Grégoire Carpentier [25] a proposé de mêler la programmation par contraintes avec de l'optimisation multi-critère pour élaborer un système d'aide à l'orchestration. Le principe consiste pour le compositeur à fournir un "son cible" enregistré, que l'on va chercher à reproduire par une combinaison de notes jouées par les instruments d'un orchestre. Une analyse du son cible et la connaissance des caractéristiques des sons produits par les instruments, permettent de se ramener à l'optimisation d'une fonction impliquant des descripteurs sonores. La programmation par contraintes est alors utilisée pour décrire les caractéristiques de l'orchestre (nombre et type d'instruments) et réduire l'espace des solutions.

Citons enfin les travaux de Jérémie Vautard [94], dont les applications musicales sont certes un "effet de bord", mais qui méritent d'être mentionnés de part leur originalité. Il s'agit ici de sonifier les traces d'exécution d'algorithmes de résolution de systèmes de contraintes. Chaque événement de la résolution (mouvement dans l'arbre de cherche, découverte d'une solution ...) est associé à un événement musical. L'objectif initial est de proposer un complément sonore aux systèmes de visualisation de traces. Mais le procédé constitue également un mécanisme de génération automatique de musique, et les pièces ainsi créées laissent percevoir des structures particulières, héritées du graphe de recherche de l'exploration.

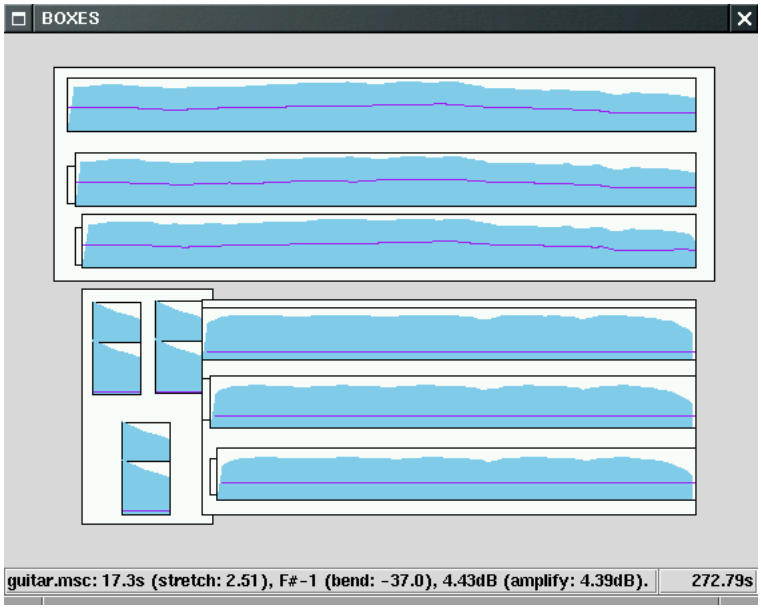


FIG. 2.8 – Une capture d'écran du logiciel *Boxes*

2.3 Conclusion

Ce panorama des formalismes utilisés en Composition Assistée par Ordinateur, nous permet d'identifier des cadres dans lesquels l'approche que cherchons à adopter pourrait se développer. Ainsi, l'approche temporelle des *data structure* de *Pure Data* qui proposent une représentation sur une ligne de temps d'événements liés à l'exécution en temps réel de processus sonore, semble s'approcher de la conception temporelle de nos partitions interprétables. Concernant l'expression de relation temporelle, l'exemple de *Boxes*, outre qu'il est le seul à développer cette possibilité aussi loin, propose un formalisme très proche de l'écriture du cadre de l'interaction au travers de contraintes sur les dates d'événements. Enfin, concernant la résolution interactive de systèmes de contraintes, la résolution par propagation de perturbations utilisées par *MusicSpace* pourrait être un exemple à suivre, dans la mesure où les valeurs des dates d'événements sur la partition écrite constituent une solution à l'ensemble des contraintes sur les dates, solution qui sera modifiée par l'interprète pendant l'exécution.

Chapitre 3

Modèles de représentation du temps

L'écriture de l'interprétation au travers des modifications agogiques, nécessite la définition de contraintes temporelles. Nous l'avons vu dans le chapitre 1, ces contraintes sont deux ordre : qualitatives et quantitatives. La possibilité de définir des relations temporelles entre des objets dépend du modèle du temps que nous adoptons. Nous cherchons donc à déterminer quels sont les modèles temporels dont nous pourrions nous appuyer pour la formalisation de notre modèle.

Nous nous intéressons ici uniqement à des modèles reliés à la musique, ou dont l'utilisation dans un contexte musical a été discuté. Nous distinguons classiquement les trois types de modèles suivants

- modèles séquenciel
- modèles hiérarchiques
- modèles logiques

Pour de plus amples détails sur les modèles du temps dans la musique, le lecteur pourra se reporter à [34].

Nous introduisons en outre l'algèbre des *S-langages* que nous utilisons par la suite.

3.1 Modèles séquenciels

Les modèles temporels séquentiels sont les modèles utilisés par les séquenceurs classiques. Le principe est ici de représenter les événements sur une ligne de temps en associant à chacun d'eux un label représentant sa date dans le référentiel temporel. On obtient alors une séquence d'événements.

Cette représentation à l'avantage d'être simple, précise et de fournir la position temporel d'un événement sans aucun calcul. Ceci lui confère des propriétés intéressantes pour l'édition, justifiant ainsi sa popularité dans les séquenceurs. Elle permet en outre d'effectuer des comparaisons entre les événements (précédence...) en manipulant directement les labels des événements. En revanche, elle est impuissante à exprimer des relations temporelles entre les événements. Si elle autorise de vérifier par exemple qu'un événement A est postérieur à un événement BB, elle s'avère incapable de maintenir cette relation pendant une phase d'édition.

3.2 Modèles hiérarchiques

Une manière de pallier aux limitations des modèles séquentiels, est de considérer des modèles hiérarchiques. Cette approche a été particulierement développée par Mira Balaban [14]. Dans e formalisme, défini à l'origine pour l'intelligence artificielle mais applicable aux situations musicales, le temps est structuré en type d'entités : les *Elementary Structured Histories (ESH)* qui sont des événements ou actions et qui disposent d'une durée, et les *Structured Histories (SH)*, qui regroupent des *ESH* et des *SH* de niveau inférieur. On a alors une représentation hiérarchisée du temps. Les positions temporelles des éléments sont alors définies, non plus de manière absolue mais relativement aux débuts des structures dans lesquelles elles sont incluses , on parle alors non plus de date mais d'estampillage (*time-stamp*). La simplicité des modèles séquenciels est ici conservée, mais tout en autorisant des constructions plus élaborées. Certaines

manipulations sont également rendues plus aisées dans ce modèle. Par exemple, modifier la date de début d'un *SH* dans le référentiel de sa structure contenante, modifie automatiquement la date des objets qu'il contient dans ce même référentiel.

Ce type de modèle est bien adaptée aux situations musicales, dans la mesure où la notation traditionnelle de la musique est largement structurée (mesures, parties...). On retrouve ce type de modèle dans *OpenMusic* avec la notion de conteneur (*container*) [12]. Le conteneur est une classe écrite donc en *CLOS*[13], dont hérite les objets musicaux d'*OpenMusic*. Cette classe possède trois attributs :

- un *onset* (o), qui représente la position temporelle de l'objet dans le référentiel de structure qui le contient
- une durée (e)
- une unité de subdivision (u) dans laquelle vont être exprimés, les onset des objets contenus dans le conteneur

C'est au travers de l'unité de subdivision u que *OpenMusic* dispose de représentations temporelles hétérogènes. Cette subdivision permet également de construire aisément des fonctions de modification proportionnelle de durée des objets, en agissant directement sur la valeur de u.

En revanche, ce modèle présente quelques inconvénients. Tout d'abord il nécessite un calcul pour obtenir la date d'un événement dans un référentiel qui n'est pas celui de sa structure contenante. De plus, si la structuration permet de définir des opérateurs de concaténation à même d'exprimer des relations temporelles, le nombre de ces relations exprimables avec ces modèles est limité.

3.3 Modèles formels

La définition d'une logique temporelle a été au centre du débat philosophique depuis ses origines, et a été particulièrement discutée par les logiciens des XIX^e et XX^e siècles.

Nous nous arrêtons ici sur un exemple d'algèbre d'intervalles, les relations dites de Allen, et sur une logique permettant de raisonner sans table de transtivité, les *S-langages*.

3.3.1 Les relations sur les intervalles

Il s'agit de relations qui permettent de décrire l'agencement temporel entre deux "événements" de durée non nulle. Pour éviter toute ambiguité avec la notion d'événement introduite dans la partie précédente, nous parlerons d'objets temporels. Ces relations sont très utilisées en intelligence artificielle, surtout depuis les travaux de J.A. Allen [4]. En tant que linguiste, Allen développa le formalisme de ses relations pour effectuer de l'analyse de discours, et déduire automatiquement une organisation temporelle globale à partir de relations locales. Les objets qu'il manipule sont donc les actions décrites dans le discours à analyser, leur seule propriété temporelle étant d'avoir un début et une fin. L'objectif est d'exhiber à partir des éléments de discours des ordres partiels entre les débuts et les fins des objets temporels. Pour représenter ces ordres partiels, Allen s'appuie sur un jeu de 13 relations de base composé de 7 relations directes et de 6 relations inverses :

$$\begin{aligned} rel = \quad & \{before, meets, overlaps, starts, during, finishes, equals \ldots \\ & \ldots beforei, meetsi, startsi, duringi, finishesi\} \end{aligned}$$

Ces relations bien que souvent appelées "relations de Allen" sont préexistantes à ses travaux, l'apport de Allen se situant dans létablissement de la table de transitivité liée à ces relations.

La figure 3.3.1 présente les 7 relations directes. Pour que la définition des relations inverses aient un sens, il est important de préciser que les ordres partiels décrits par les relations impliquent des inégalité

[13] Common Lisp Object System

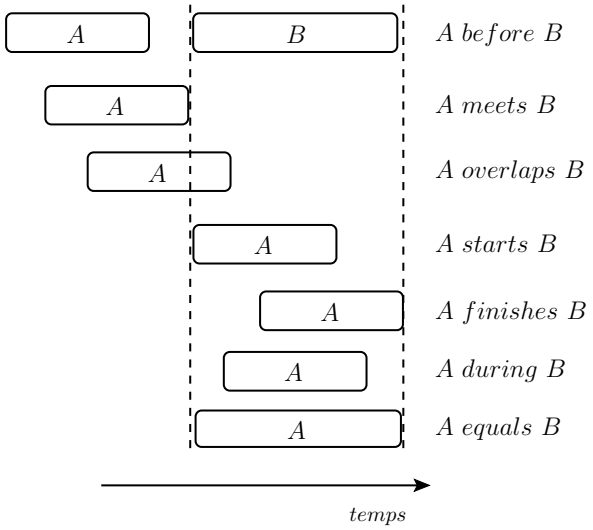


FIG. 3.1 – Les relations de Allen

strictes :

$$\begin{array}{lcl}
\text{A before B} & : & end(A) < start(B) \\
\text{A meets B} & : & end(A) = start(B) \\
\text{A overlaps B} & : & start(A) < start(B) \\
 & & start(B) < end(A) \\
 & & end(A) < end(B) \\
\text{A starts B} & : & start(A) = start(B) \\
 & & end(A) < end(B) \\
\text{A during B} & : & start(B) < start(A) \\
 & & end(A) < end(B) \\
\text{A finishes B} & : & start(B) < start(A) \\
 & & end(A) = end(B) \\
\text{A equals B} & : & start(B) = start(A) \\
 & & end(A) = end(B)
\end{array}$$

Notons qu'est implicitement admise la relation ($start(A) < end(A)$) qu'implique la durée non nulle d'un objet A.

Concernant les relations inverses, elles se déduisent simplement des relations directes :

Propriété 3.1 *Soient* A *et* B*, deux objets temporels, "*r*" une relation directe et "*ri*" sa relation inverse :*

$$A\ r\ B \Leftrightarrow B\ ri\ A$$

La relation *equals* étant sa propre relation inverse, on obtient bien les 13 relations sus-citées.

Allen développe alors une algèbre temporelle en intoduisant l'opérateur logique *ou* et en autorisant les combinaisons entre les relations. On peut alors décrire les relations entre deux objets temporels sous la forme de disjonctions des relations de base. Allen propose enfin une table de transitivité permettant de déduire les relations temporelles non-exprimées à partir des relations présentes dans le discours. Nous reproduisons cette table sur la figure 3.2.

La simplicité des relations de base et la puissance d'expression offerte par la grammaire ont assuré aux relations de Allen un succès certain. De nombreux travaux autour du temps s'appuient sur leur formalisme,

	<	**>**	**d**	**di**	**o**	**oi**	**m**	**mi**	**s**	**si**	**f**	**fi**
before **<**	<	<> d di o oi m mi s si f fi	< o m d s	<	<	< o m d s	<	< o m d s	<	<	< o m d s	<
after **>**	<> d di o oi m mi s si f fi	>	> oi mi d f	>	> oi mi d f	>	> oi mi d f	>	> oi mi d f	>	>	>
during ***d***	<	>	d	<> d di o oi m mi s si f fi	< o m d s	> oi mi d f	<	>	d	< oi mi d f	d	< o m d s
duringi ***di***	< o mi di fi	> oi id mi si	o oi d di s si f fi =	di	o di fi	oi di si	o di fi	oi di si	o di fi	di	oi di si	di
overlaps ***o***	<	> oi di si mi si	o d s	< o m di fi	< o m	o oi d di s si f fi =	<	oi di si	o	di fi o	d s o	< o m
overlapsi ***oi***	< o m di fi	>	oi d f	> oi mi di si	o oi d di s si f fi =	> oi mi	o di fi	>	oi d f	oi > mi	oi	oi di si
meets ***m***	<	> oi mi di si	o d s	<	< o d s	<	f fi =	m	m	d s o	<	d s o
meetsi ***mi***	< o m di fi	>	oi d f	>	oi d f	>	s si =	>	d f oi	>	mi	mi
starts ***s***	<	>	d	< o m di fi	< o m	oi d f	<	mi	s	s si =	d	< m o
startsi ***si***	< o m di fi	>	oi d f	di	o di fi	oi	o di fi	mi	s si =	si	oi	di
finishes ***f***	<	>	d	> oi mi di si	o d s	> oi mi	m	>	d	> oi mi	f	f fi =
finishesi ***fi***	<	> oi mi di si	o d s	di	o	oi di si	m	si oi di	o	di	f fi =	fi

FIG. 3.2 – Table de transitivité de Allen pour les relations d'intervalles

parmi les nombreuses applications notamment en intelligence artificielle. Dans le cadre de l'informatique musicale, ces relations ont été utilisées dans l'environnement de composition assistée par ordinateur *Boxes*, afin de proposer une approche de la composition basée sur les rapports temporels qu'entretiennent deux à deux les objets musicaux, ces rapports conduisant à une organisation temporelles globale et des dates d'événements calculées par le système.

Parrallèlement à l'algèbre d'intervalles, s'est développée une algèbre de points, impliquant des événements de durée nulle. Cette algèbre manipule trois relations :

- précédence
- synchronicité
- postériorité

toujours avec une table de transitivité.

Vilain [95] a ensuite unifié ces deux types d'algèbres. D'autres algèbres faisant intervenir des structures plus complexes ont été proposées. Celles-ci s'appuient également sur des tables de transitivité.

3.3.2 S-langages

Le formalisme des S-langages visent à développer un calcul d'ordonnancement temporel sans tables de transitivité, basée sur les langages formels. Ce formalisme a été introduit par Sylviane Schwer [47].

L'idée est de représenter les événements par des lettres, les pré-ordres sur ces événements par des mots et des ordres partiels (ensembles d'ordres possibles) par des langages. Des opérateurs spécifiques sont alors définis sur ces langages pour raisonner temporellement.

Définition 3.1 *Soit X un alphabet, un S-alphabet sur X est un sous ensemble non-vide de $2^X - \emptyset$. Un élément d'un S-alphabet est une S-lettre. Un mot sur un S-alphabet est un S-mot et un ensemble de S-mots est un S-langage.*

Un S-alphabet est habituellement désigné par $\hat{X} = 2^X - \emptyset$. Un singleton est représenté par son unique lettre. Nous écrirons les S-lettres horizontalement : $\widehat{\{a,b\}} = \{a, b, \{a, b\}\}$.

Les lettres d'un alphabet X représentent des événements, les S-lettres représentent soient une occurence d'un événement (singleton), soit les occurences simultanées de plusieurs événements. Un S-mot représente alors un pré-ordre entre des occurences d'événements.

Définition 3.2 *Soient $X = \{x_1, \ldots, x_n\}$ un alphabet et $f \in \hat{X}^*$. $\| f \|_x$ désigne le nombre d'occurences de la lettre x de X dans les S-lettres de f, et*

$$\| f \| = \sum_{i=1}^{n} \| f \|_{x_i}$$

. Le vecteur de Parikh *de f, noté $\overrightarrow{f}$, est le n-uplet $(\| f \|_{x_1}, \ldots, \| f \|_{x_n})$.*

A titre d'exemple : le S-mot $f = \{a, b, \{b, c\}, \{a, b, c\}, c, c\}$ est telle que :

$$\overrightarrow{f} = (2, 3, 4)$$

.

Définition 3.3 *L'alphabet d'un S-mot f, désigné par $\alpha(f)$ est l'ensemble des lettres apparaissant dans f, tandis que l'alphabet de Parikh de f est le couple $< \alpha(f), \overrightarrow{f} >$. L'ensemble des S-mots partageant un un même alphabet de Parikh β est appelé un* S-universe *désigné par $\mathscr{U}(\beta)$.*

Le S-univers représente tous les ordres possibles entre un ensemble donné d'événements.

Un S-langage peut être décrit soit par l'ensemble de ses mots lorsque celui-ci est fini, soit par des expresssions sur des S-langages. Ces expressions sont alors appelées des *S-expressions*.

Comme les S-languages sont des langages formels, il est possible de défnir sur eux les opérateurs des langages tels que l'*union*, l'*intersection*, la *concatenation*... De plus,on utilise deux opérateurs spécifiques : la *S-projection* et la *jointure*.

La S-projection d'un S-mot $f \in \hat{X}^*$ sur un sous-alphabet $Y \subset X$ est noté $f|Y$ et correspond au S-mot obtenu à partir de f en retirant toutes les occurrences of lettres qui ne sont pas dans Y. En reprenant l'exemple précédent de f, on a : $f_{|\{a,b\}=\{a,b,\{a,b\}\}}$.

La projection et la jointure subsument d'une certaine manière les tables de transitivité des algèbres temporelles, elle permet de déduire un ordre sur un sous-ensemble d'événements, induit par un ordre sur tous les événements.

L'opération de jointure est la plus importante.

Définition 3.4 *Soient $f \in \hat{X}^*$ et $g \in \hat{Y}^*$, jointure de f et g est le S-langage $f \bowtie g = \{h \in \hat{X_f}^* \cup \hat{Y_g}^*, h_{|X_f} = f$ et $h_{|Y_g} = g\}$.*

La jointure permet de calculer tous les ordres sur un ensemble d'événements, dont on peut déduire des ordres donnés sur deux sous-ensembles d'événements. C'est dons une opération très puissante.

On peut généraliser la jointure de deux langages par :

$$L \bowtie L' = \bigcup_{f \in L, g \in L'} f \bowtie g$$

Le lien entre la projection et la jointure, et les tables de transitivités des algèbres temporelles s'expriment de la manière suivante :

$$R(a,c) = \Pi_{a,c} R(a,b) \bowtie R(b,c)$$

où l'opération R renvoie le S-mot représentant le pré-ordre entre deux événements et Π dénote la S-projection. C'est ce calcul dans l'algèbre des S-langages qui permet de se passer de table transtivité.

3.4 Représentations graphiques et automates

Les modèles de représentation du temps ont également connu des versions graphiques. Nous en rappelons ici certaines dont nous faisons usage dans la suite.

3.4.1 Diagrammes de Hasse et graphes d'événements

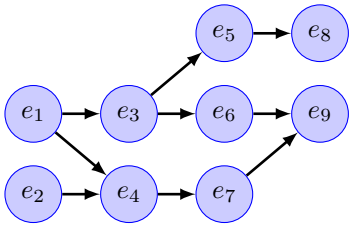
Les diagrammes de Hasse doivent leur nom au mathicien allemand Helmut Hasse et permettent de représenter des ensembles finis disposant d'une relation d'ordre. Il s'agit d'un graphe dont les sommets représentent éléments de l'ensemble et les arrêtes, les relations deux à deux entre les élements. La manière de dessiner le graphe à son importance : en supposant une relations d'ordre notée R et deux éléments de l'ensemble a et b,si aRb, alors le sommet représentant b sera dessiné plus haut que celui représentant a. Cette convention induit une orientation verticale générale du diagramme et évite d'avoir à orienter les arêtes. Une arrête existe entre deux sommets si les élément représentés par ces sommets sont en relation. Enfin, seules les relations constituant la cloture par transitivité de l'ordre sont représentés, ce qui limite le nombre d'arrêtes dans le graphe. Il est possible de représenté des ensembles d'événements temporellement ordonnés, en s'appuyant par exemple sur une relations de précédence.

Nous présentons sur la figure 3.4.1 l'ensemble $\{e_i\}_{i\in[1,9]}$ d'évenements, dont le pré-ordre temporel s'exprime par le S-langage suivant :

$$\{e_1 e_3 [e_5 e_8 \otimes e_6 e_9]\} \bowtie \{[e_1 \otimes e_2] e_4 e_7 e_9\} \tag{3.1}$$

La représentation de la synchronisation de deux événements dans un diagramme de Hasse, pose soucis car elle implique la représentation de deux événements synchrones par le même sommet du graphe.

Ce problème est absents de la représentation par "graphes d'événements". Ces graphes, spécialement conçus pour la représentation d'ensembles d'événements temporels ordonnés, fonctionnent sur le même principe que les diagramme de Hass avec relation de précédence. Cependant, il n'y a pas d'orientation générale du graphe mais une orientation des arcs. Ainsi, il existe un arc du sommet a vers le sommet b si l'événement a précède l'événement b. La synchronisation s'exprime alors par une double précédence. Nous présentons sur la figure 3.4.1 le graphe d'événement représentant le même pré-ordre temporel que le diagramme de Hasse.



(a) Graphe d'événements

FIG. 3.3 – Le diagramme de Hasse et le graphe d'événements associés au S-langage de l'expression 3.1

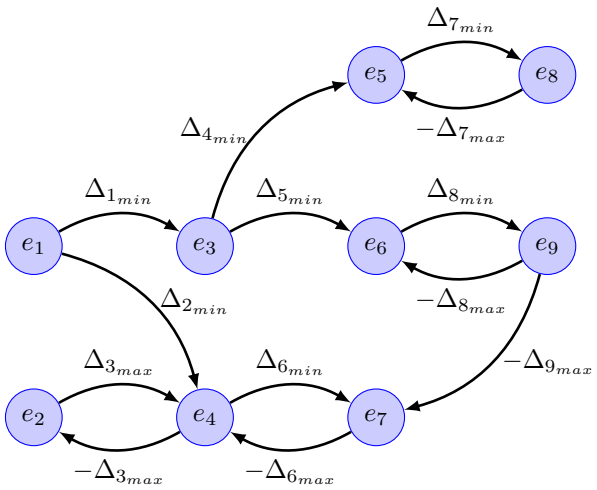


FIG. 3.4 – Le graphe AOA représentant le même ensemble que les graphes de la figure 3.3 associé à des valeurs d'intervalles arbitraires

3.4.2 Graphes AOA

Les diagramme de Hasse et les graphes d'événements permettent des représentations impliquant uniquement des relations qualitatives entre les événements. D'autres représentation ont été développée pour représenter des relatiosn quantitatives entre événements. C'est le cas des graphes AOA [] (*activity-on-arc graph*). Ceux-ci reprènent le principe des graphes d'événements ajoutant des informations conernant les intervalles de temps séparant les événements, sous formes d'étiquettes sur les arcs du graphe. En notant $d(e)$ la date d'un événement e, si :

$$\leq 0 \Delta_{min} \leq d(e_2) - d(e_1) \leq \Delta_{max}$$

alors, il existe un arc de sommet représentant e_1 vers celui représentant e_2 étiqueté avec la valeur Δ_{min}. Et si $\Delta_{max} \neq +\infty$, alors il existe un arc du sommet représentant e_2 vers celui représentant e_1 étiqueté avec la valeur $-\Delta_{max}$.

Les graphes AOA permettent, comme les graphes d'événements, de déduire des relations par transitivité. On peut ainsi déduire les valeurs possibles pour un intervalle entre deux événements. C'est de cette manière que nous nous en servons dans la suite du mémoire.

3.4.3 Automates et Réseaux de Petri

Les représentations exposées précédemment ne disposent pas de machine d'exécution permettant de simuler l'ordonancement d'un ensemble d'événements temporels, ou de construire directement dont les

propriétés temporelles sont spécifiées au préalable.

C'est le cas d'autres représentations spécialement concues pour décrire les liens temporelles entre les étapes de l'exécution d'une machine abstraite. Deux approches alors sont possibles :

– l'approche "états"
– l'approche "événements"

L'approche "états" est celle adapotée par les automates. Elle consiste à représenter les états par lesquels va passer la machine au cours de son exécution. Les changements d'états possibles sont représentés par des transitions entre états. Une transition correspond au déclenchement d'un événement. Le formalisme d'automates temporisés permet de préciser des données temporelles pour ces transitions. La limite classique de l'approche par états est la difficulté qu'elle a à représenter des exécutions massivement parrallèles. En effet, une grande incertitude sur l'ordre temporel d'un ensemble d'événements conduit à une explosion du nombre d'états de l'automate.

L'approche "événements" évite cet eccueil en laissant en arrière plan la représentation des états globaux du système et en se focalisant sur ses changements d'état. C'est l'approche adopté par les réseaux de Petri. Comme une partie de notre problème concerne la question de l'exécution des partitions, nous sommes intéressé par des représentation d'ensemble d'événements temporels ordonnés disposant d'une machine d'exécution. De plus, les partitions pouvant potentiellement présenter des parties massivement parallèles, nous nous sommes tournés vers les réseaux de Petri. Pour permettre une lecture aisée des parties suivantes, nous rappelons ici quelques notions de base sur ces réseaux. Une structure particulière de Petri est plus spécifiquement dédié à la représentation des ensembles d'événements temporels ordonnés. Comme nous nous appuyons fortement sur ces derniers, nous leurs consacrons le chapitre suivant ainsi qu'aux liens qu'on peut faire avec les S-langages.

3.5 Quelques définitions du formalisme des réseaux de Petri

Pour une présentation plus complète de la théorie des réseaux de Petri, nous invitons le lecteur à se reporter à [42] et [43]. Les notations et définitions de base que nous employons sont d'ailleurs issues de ces deux ouvrages.

3.5.1 Réseaux de Petri Places-Transitions

Les réseaux de Petri les plus simples et qui servent de base à tous les autres formalismes sont ceux de la classe des réseaux *places-transitions* introduits par C. A. Petri dans son travail de thèse en 1962 [79, 60]. Les réseaux de Petri furent introduits pour permettre la communication et la synchronisation entre automates.

Formellement, il s'agit de graphes bi-parties dont les 2 types de nœuds s'appellent *places* et *transitions*.

Définition 3.5 *Un réseau de Petri* place-transition *Petri est un 4-uplet :*

$$R = (P, T, Pre, Post)$$

ou :

- $P = \{p_1, \ldots, p_n\}$, *un ensemble de places*
- $T = \{t_1, \ldots, t_n\}$, *un ensemble de transitions*
- $Pre : P \times T \to \mathbb{N}$, *une application d'incidence* avant
- $Post : P \times T \to \mathbb{N}$, *une application d'incidence* arrière

Le sens des applications d'incidence est le suivant :
– *Pre(p,t) est le poids associé à l'arc (p,t)*
– *Post(p,t) est le poids associé à l'arc (t,p)*
Un poids d'arc nul correspond naturellement à une absence d'arc

Les arcs des réseaux sont représentés au travers de poids définis entre des nœuds, à la manière de la représentation des automates sous formes de matrices.

Pour la clarté des propos, il nous faut nommer les ensembles de prédécesseurs et de successeurs d'un nœud.

Définition 3.6 *Soit R un réseau de Petri et n un nœud de R ($n \in P \cup T$) :*

- *${}^\bullet n$ représente l'ensemble des prédécesseurs de n*
- *$n^\bullet$ représente l'ensemble des successeurs de n*

De plus nous noterons ${}^*n$ et n^* les clotures par transitivité de ces opérateurs.

L'état global d'un réseau de Petri est représenté par une répartition de jetons au travers de ses places.

Définition 3.7 *Un réseau de Petri* marqué *est un couple :*

$$PN = (R, m)$$

ou :

- *R=(P,T,Pre,Post) est un réseau de Petri* places-transitions
- *$m : P \rightarrow \mathbb{N}$ est une distribution de jetons dans les places appelée* marquage

Nous noterons $m(R)$ le marquage d'un réseau R et $m(p)$ ou m_p le marquage de la place p correspondant, c'est à dire le nombre de jetons contenus dans la place p. De plus, nous noterons m_0 le marquage initial qui donne la valeur initiale du nombre de jeton dans toutes les places et donc l'état global initial du système.

L'évolution d'un réseau s'effectue au travers des franchissement des transitions. Le franchissement d'une transition constitue un changement d'état global du système.

Définition 3.8 *Soit un réseau de Petri marqué $PN =< R, m >$, une transition t de R est dite* sensibilisée *par m (*franchissable *ou* tirable *dans PN) ssi :*

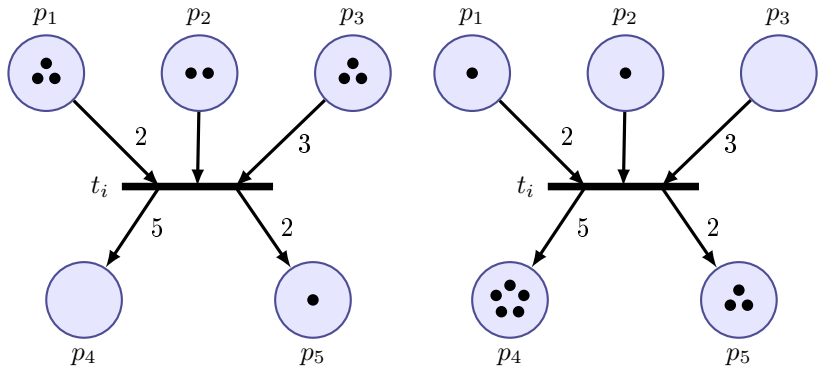
$$\forall p_i \in P, m(p_i) \geq Pre(p_i, t)$$

La règle de sensibilisation indique qu'une transition est franchissable si chacun de ses prédécesseurs contient un nombre de jetons au moins égal au poids de l'arc menant à la transition.
Nous noterons la sensibilisation d'une transition t par un marquage m par : $m[t >$.

Quand une transition est franchissable, elle peut être franchie et le changement d'état induit correspond à une consommation de jetons en amont de la transition et une production en aval.
Il est important de noter que la quantité de jetons produite en sortie est indépendante de celle consommée en entrée comme sur la figure 3.5.

3.5.2 Le temps dans les réseaux de Petri

La théorie des réseaux de Petri a connu plusieurs extensions permettant de modéliser des problèmes spécifiques. Une branche importante de la théorie consiste à introduire la notion de temps dans les réseaux. La première temporisation des réseaux de Petri fut la formalisation des réseaux de Petri *temporels*. Introduits par Merlin en 1974 [70], ces réseaux font intervenir le temps en autorisant des délais entre la sensibilisation d'une transition et son franchissement.
Les réseaux de Petri temporels s'avèrent incapables de modéliser simplement des systèmes dans les lequel des processus autonomes (asynchrones) doivent se synchroniser. En effet, les réseaux de Petri temporels considèrent que seuls les "événements" (franchissements de transition) sont dignes de porter



(a) Une transition sensibilisée (b) Le tir de cette transition

FIG. 3.5 – Un exemple de tir d'une transition sensibilisée

des informations temporelles, alors que les systèmes dans lesquels des objets autonomes doivent se synchroniser, il est nécessaire d'associer des intervalles de temps à certains "états" (l'exécution des objets). Cependant, les contraintes temporelles imposées à l'exécution des objets en question ne sont pas nécessairement les mêmes pour chacun d'entre eux. On parle alors de *composition temporelle* pour désigner les règles de synchronisation des exécutions d'objets au niveau des transitions.

Les réseaux de Petri *à flux autonomes* ont été introduits ([38]) pour permettre la composition temporelle dans les réseaux. Ces réseaux tirent leur nom de l'assimilation des processus autonomes à des flux.

Définition 3.9 *Un réseau de Petri à flux autonomes (encore appelé réseau de Petri temporel à flux) est un triplet*

$$RdPFA =< R, Ita, Sem >$$

où :

- *R est un réseau de Petri* places-transitions
- $Ita : A \rightarrow \mathbb{Q}^+ \times \mathbb{Q}^+ \times (\mathbb{Q}^+ \cup \{\infty\})$
- $Sem : T \rightarrow \{semantiques\}$

Dans cette définition, A est l'ensemble des arcs reliant une place à une transition i.e :

$$A = \{a = (p,t), p \in P, t \in T | Pre(p,t) > 0\}$$

Ita permet d'associer à chaque arc 3 valeurs temporelles :
- *une date de tir au plus tôt*
- *une date de tir nominale*
- *une date de tir au plus tard*

Chaque arc est donc associé à un intervalle de tir et une valeur nominale. Ces valeurs sont à interprétées relativement à la date à laquelle il y a suffisamment de jetons dans la place origine de l'arc pour permettre la tir de la transition.

Dans le cas d'une transition disposant de plusieurs arcs entrant, il faut combiner les diférents intervalles de tir pour déterminer une plage absolue de date de tir de la transition.

Pour une transition donnée, la règle de combinaison des intervalles des arcs est définie par la sémantique qui lui est associée.

L'ensemble de sémantiques contient 10 règles de tir d'une transition qui représente 10 possibilités de synchronisation temporelle à savoir : Et-Pur, Et, Et-And, Ou, Ou-fort, Maître, Et-Maître, Ou-Maître, Maître-fort, Maître-faible

Arrêtons-nous sur quelques-unes des sémantiques de tir dont nous userons par la suite.

Sémantique du Et-Pur

Soit t_j une transition, A_j est l'ensemble des arcs entrants de t_j i.e. :

$$A_j = \{a_i = (p_i, t_j), p_i \in P | Pre(p_i, t_j) > 0\}_{i \in [1,l]}$$

On note pour $a_i \in A_j$:
– $ita(a_i) =< \alpha_i, n_i, \beta_i >$
– τ_i la date telle que $M(p_i) \geq Pre(p_i, t_j)$

Définition 3.10 *Une transition t_j avec une sémantique Et-Pur est tirable à la date absolue τ^f si et seulement si pour tout a_i de A_j les deux conditions suivantes sont vraies :*

- *t_j est sensibilisée à la date τ^f :*
$$\forall p_i, M(p_i) \geq Pre(p_i, t_j)$$
- *τ^f est telle que :*
$$\forall a_i, (\tau_i + \alpha_i) \leq \tau^f \leq (\tau_i + \beta_i)$$

On a donc $MIN \leq \tau^f \leq MAX$ avec :

$$MIN = \max_{i \in [1,l]} \{\tau_i + \alpha_i\}$$

$$MAX = \min_{i \in [1,l]} \{\tau_i + \beta_i\}$$

Sémantique du Ou

En reprenant les mêmes notations que pour la sémantique du *E*t-Pur la sémantique du *O*u s'exprime de la manière suivante.

Définition 3.11 *Une transition t_j de type* Ou *est tirable à la date absolue τ^f, si et seulement si t_j est sensibilisée à la date τ^f et $\tau^f \in [MIN, MAX]$ avec :*

$$MIN = \min_{i \in [1,l]} \{\tau_i + \alpha_i\}$$

$$MAX = \max_{i \in [1,l]} \{\tau_i + \beta_i\}$$

Et-Maître

Certaines sémantiques s'appuient sur un arc particulier auquel elles donnent une importance particulière par rapport aux autres. Ces sémantiques sont appelées sémantiques *Maître*. Nous présentons ici celle du *Et-Maître*.

Définition 3.12 *Une transition t_j de type* Et-*Maître définie sur l'arc $a_k = (p_k, t_j)$, est tirable à la date absolue τ^f si et seulement si t_j est sensibilisée à la date τ^f et $\tau^f \in [MIN, MAX]$ avec :*

$$MIN = \max_{i \in [1,l]} \{\tau_i + \alpha_i\}$$

$$MAX = \min(\tau_k + \alpha_k, \max_{ß \in [1,l]} \{\tau_i + \alpha_i\})$$

La figure 3.6 présente une transition d'un réseaux de Petri à flux autonomes ainsi qu'une comparaison entre les intervalles de tir de cette transition pour les trois sémantiques que nous avons présentées.

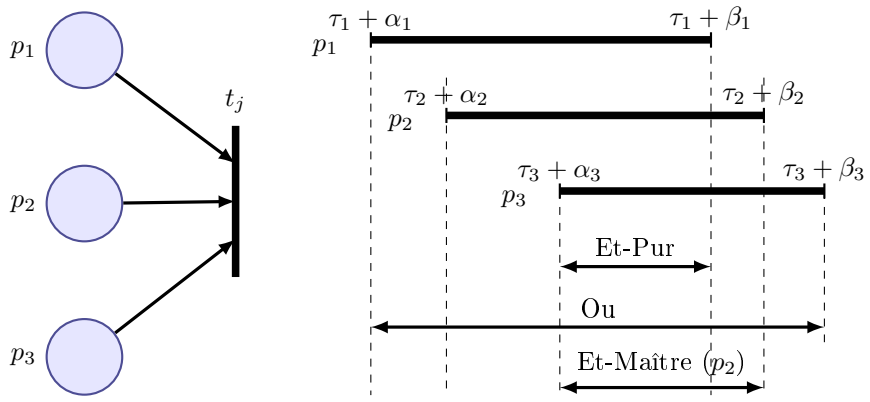


FIG. 3.6 – Une transition d'un réseau temporel à flux

3.5.3 Réseaux de Petri Hiérarchisés à flux temporels

En complément des 10 sémantiques de synchronisation "parallèle" des réseaux de Petri temporels à flux autonomes, a été ajoutée une sémantique de synchronisation "séquentielle". Il ne s'agit plus de permettre la synchronisation de processus asynchrones en un point donné, mais de composer une succession de processus en un seul, en déduisant des propriétés temporelles des processus de la séquence, celles du processus résultant de la composition.

L'expression de cette sémantique est très simple, nous la présentons directement exprimée avec des arcs de réseaux à flux autonomes.

Définition 3.13 *Soient a_i et a_j deux arcs d'un RdPFA avec $ita(a_i) =< \alpha_i, n_i, \beta_i >$ et $ita(a_j =< \alpha_j, n_j, \beta_j >$, on a alors :*

$$ita(seq(a_i, a_j)) =< \alpha_i + \alpha_j, n_i + n_j, \beta_i + \beta_j >$$

Armé des sémantiques séquentielles et parallèles, ont été définies des règles de réduction des réseaux de Petri temporels à flux. Il s'agit de composer récursivement les séquences de transitions et les synchronisations parallèles pour aboutir à un unique arc. Naturellement, la réduction totale en un seul arc n'est possible que si la structure du réseau ne contient que des configurations séquentielles et parallèles. Les réseaux disposant d'une telle structure ont été formalisés avec la notion de réseaux *structurés*.

Définition 3.14 *Un réseau de Petri structuré à flux temporels est un réseau de Petri temporel à flux autonomes construit par récursion de schémas de synchronisation séquentiels et parallèles.*

Du point de vue des graphes, la structure d'un réseau structuré est celle d'un graphe série/parallèle [45]. La réduction des séquences à des arcs en appliquant les règles de sémantiiques définit une relation d'équivalence temporelle dans l'ensemble des réseaux structurés. Deux réseaux étant équivalents si ils se réduisent au même arc.

Tout l'intérêt des réseaux structurés à flux temporels, réside dans la possibilité de déduire des propriétés temporelles "globales" à partir des propriétés temporelles des arcs qui le composent. Intuitivement, ces propriétés globales sont définies entre un "début" et une "fin" du réseau. A partir de là pourquoi ne pas offrir une représentation de réseaux "complexes" par encapsulation successives de sous-réseaux ? Puisqu'il est possible de représenter un réseau comme un processus simple aux propriétés temporelles identifiées, on pourrait tout à fait présenter des réseaux avec des niveaux de raffinement de plus en plus précis, le détails des propriétés temporelles étant remplacés par un unique arc au niveau hiérarchique supérieur. C'est précisément l'approche adoptée par les réseaux de Petri *hiérarchisés* à flux temporels ([76]).

Sans donner ici formellement la définition de ces réseaux, on peut noter qu'ils effectuent un typage des places selon les deux types :

– atomique

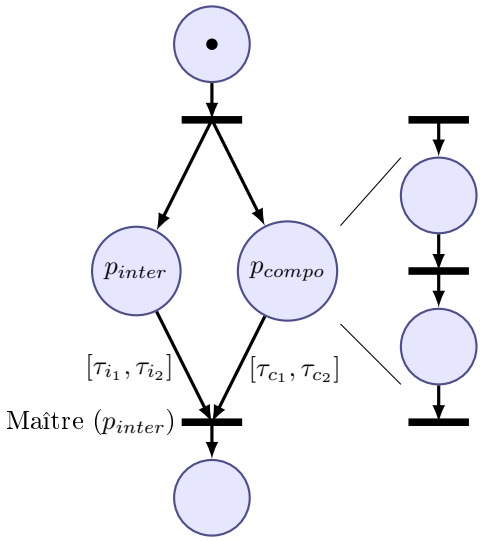


FIG. 3.7 – Modélisation d'une interruption dans les réseaux temporels à flux

– composite

Une place composite contient un sous-réseau dont l'exécution sera déclenchée lors de la production d'un jeton dans la place. Une place atomique ne contient pas de sous-réseaux.

Les règles de tirs dans les réseaux hiérarchisés prennent en compte la présence de places composites parmi les prédécesseurs d'une transtion. Les sous-réseaux inclus dans ces places composites disposent de "place de sortie" dans lesquelles la présence d'un jeton conditionne le franchissement de la transition. L'influence de ces place de sorties sur le franchissement dépend de la sémantique de la de la transition.

En notant $Fso(p)$ les places de sorties du sous-réseau inclus dans une place composite p et AM l'arc maître d'une transition de sémantique *Et-Maître*, les conditions de franchissement d'une transition t sont :

– t prise dans le réseau de Petri correspondant à son niveau hiérarchique doit satisfaire les conditions classques de tir dans les réseaux de Petri temporel à flux.
– de plus, les places composites qui précèdent t doivent remplir une condition qui dépend de la sémantique de t :
  – $sem(t) = et \Rightarrow \forall p \in {}^\bullet t, m(Fso(p)) \neq 0$
  – $sem(t) = ou \Rightarrow \exists p \in {}^\bullet t, m(Fso(p)) \neq 0$
  – $sem(t) = maitre, AM(t) = (p, t) \Rightarrow m(Fso(p)) \neq 0$

Outre la modélisation de systèmes par encapsulation, les réseaux hiérarchisés offrent un avantage certain pour la représentation des *interruptions*. La règle de tir d'une transition a en effet ceci de confortable, qu'elle arrête l'évolution d'un sous-réseau précédent une transition sans autre forme de procès. Et surtout sans préjuger de l'état interne du sous-réseau (sensibilisation des transitions, intervalles de tir ...). Simuler ce type de comportement dans un réseau non hiérarchisé avec arcs inhibiteurs par exemple, comme dans [98], s'avère bien plus complexe. Ainsi, une modélisation classique des interruptions utilise une transition de type *Maître*, pour synchroniser un arc issu d'une place composite représentant un processus d'un coté avec un arc issu d'une place représentant un processus d'interruption, la priorité de synchronisation étant donnée à l'arc d'interruption comme sur la figure 3.7.

3.6 Réseaux de Petri synchronisés

Les réseaux de Petri sont avant tout des outils d'analyse, leur utilisation "classique" consiste donc à modéliser un système donné à l'aide de réseaux, et d'utiliser les "puissants" résultats théoriques sur ces derniers pour décrire les propriétés du système.

Cependant, la structure de graphe des réseaux et leurs règles de tir en font des objets simples à

implémenter. Naturellement l'exécution d'un réseau respecte les propriétés de ce réseau. Mais baser un système sur l'exécution d'un réseau de Petri, implique que celui-ci soit capable de réagir à des événements extérieurs pour permettre des interactions d'utilisateurs avec le système. L'interaction la plus simple consiste à piloter l'évolution du réseau et donc de conditionner le franchissement de certaines transitions par l'arrivée d'événements extérieurs ; les réseaux de Petri *synchronisés* ont été introduits en 1978 par Moalla and al. [72] précisément dans ce sens.

Définition 3.15 *Un réseau de Petri synchronisé est un triplet*

$$PNS = (PN, E, Sync)$$

où :

- *PN est un réseau de Petri*
- $E = \{e_1, e_2, \ldots, e_n\}$ *est un ensemble d'événements*
- $Sync : T \rightarrow E \cup \mathbf{e_0}$ *est une application qui associe un événement à chaque transition.*

Les règles de tir classiques restent valables pour les réseaux synchronisés et seules les transitions franchissables peuvent être tirées. Par conséquent, si une transition est franchissable et que son événement associé se produit, elle est franchie ; si un événement se produit alors que la transition à laquelle il est associé n'est pas franchissable, rien ne se produit. L'événement permanent e_0 sert à homogénéiser les transitions du réseau. Toutes les transitions attendent un événement, celles qui ne sont pas déclenchées par un événement extérieur sont associées à l'événement permanent qui se produit tout le temps.

3.7 Réseaux de Petri et Musique

Les techniques de modélisation de systèmes complexes grâce aux réseaux de Petri ont rapidement conduit à des tentatives de représentation de la musique avec ce type d'outils. D'une certaine manière, on peut y voir une évolution des recherches visant à proposer des langages formels pour la représentation musicale.

3.7.1 L'école italienne

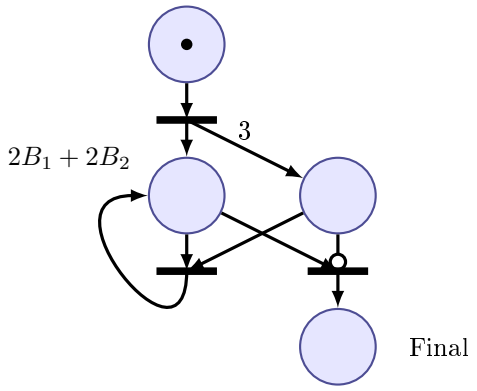
L'approche langage formel est celle adoptée au tournant des années 80-90, par l'équipe du LIM[14] formée entre autres de Goffredo Haus, Antonio Rodrigez et Antonio Camurri. Les efforts de cette équipe ont porté sur la définition d'un langage symbolique directement utilisable par les compositeurs, permettant de décrire des structures logiques difficilement exprimables par une notation classique. Les réseaux de Petri leur ont paru un outil parfaitement approprié.

Dans [55], Haus et Rodrigez présentent les bases théoriques de leur utilisation des réseaux ainsi que les premières applications réalisées. Leur objectif est d'obtenir une représentation symbolique de la musique combinant à la fois la représentation des objets musicaux eux-mêmes et les relations logiques et temporelles qui existent entre eux au sein d'une partition ; ceci à des fins de composition, d'analyse, de synthèse et de jeu.

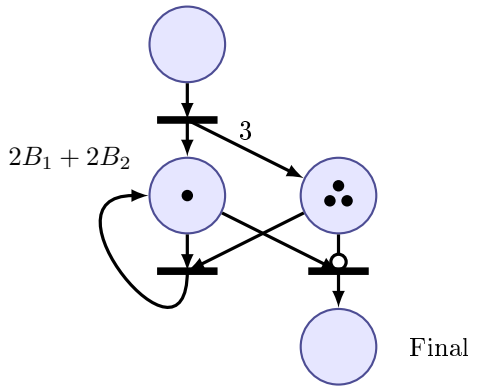
Les réseaux de Petri qu'ils se proposent d'utiliser ont les caractéristiques suivantes :

- réseaux *auto-modifiants* [93], dans lesquels le poids d'un arc (le nombre de jetons produits ou consommés lors du franchissement d'une transition) s'exprime comme une combinaison linéaire du marquage courant des places du réseau. Le poids des arcs est donc variable au cours de l'évolution du réseau. De plus, le terme d'auto-modifiant implique également que les réseaux peuvent transformer leur propre structure et leur contenu musical au cours de l'exécution.
- réseaux à arcs *inhibiteurs*, dont certains arcs empêchent le franchissement d'une transition si un jeton est présent dans la place dont ils sortent.
- réseaux *Conditions-Evénements + Places-transitions* qui mêlent la première forme de réseau de Pétri dans laquelle le marquage d'une place est un booléen avec les réseaux places-transitions.

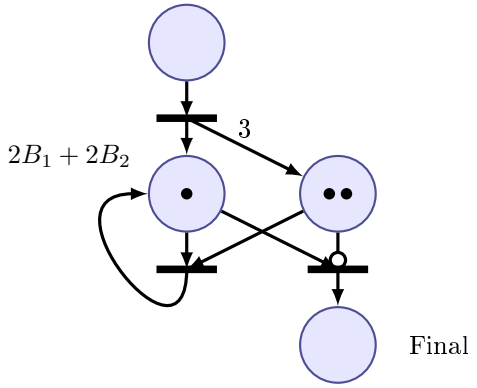
[14] Laboratorio di Informatica Musicale



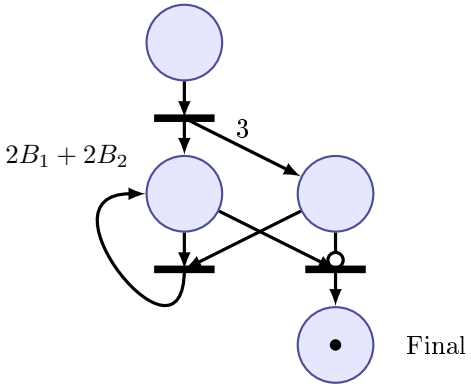
(a) État initial



(b) Après le franchissement de première transition, les jetons permettant le nombre d'itérations voulues sont produits



(c) Après la première itération



(d) État final

FIG. 3.8 – Réseaux de Petri représentant l'itération des thèmes principaux du *Boléro* de Ravel

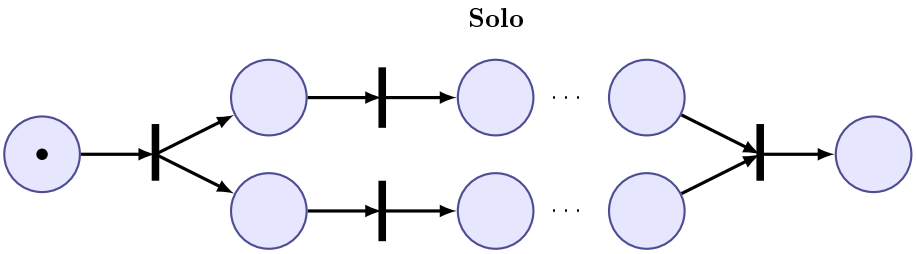
– réseaux *temporisés* [84] (à ne pas confondre avec les réseaux temporels cités plus haut) dans lesquels une durée de tir est associée à chaque transition.
– réseaux *structurés* disposant d'une forme de hiérarchie.

Lors de l'exécution d'un réseau, des effets de bord sont associés à la production de jetons dans certaines places, par exemple l'écriture dans un fichier. L'exécution d'un réseau génère donc automatiquement une partition dont la structure est directement liée à celle du réseau et au matériau musical utilisé. Comme l'exécution d'un réseau est sujette à l'indéterminisme en présence de places partagées, un même réseau représente une famille de partitions, chacune accessible par une exécution particulière du réseau. Notons au passage, que les effets de bord dépendent à la fois du contenu musical du réseau mais également de l'état interne du réseau (marquage...) au moment de leur production, le nombre de partitions que l'on peut générer à partir d'un réseau est donc très vaste.

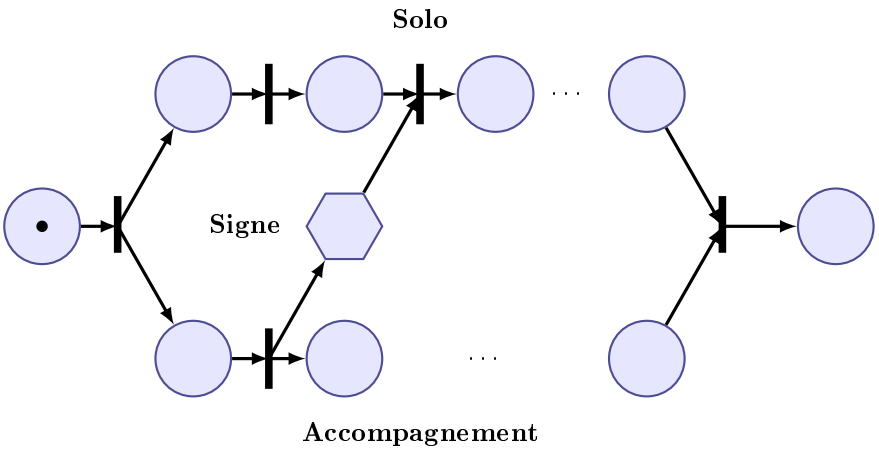
Configurations particulières

Comme les partitions générées par l'exécution d'un réseau ont des structures directement liées à celle du réseau, les auteurs ont cherché à traduire certaines situations musicales classiques en configurations dans les réseaux de Petri.

La première est la notion d'*itération* qui se simule aisément à l'aide du poids des arcs. La figure 3.7.1.0 présente l'exemple devenu célèbre du *Boléro* de Maurice Ravel, utilisé dans l'article pour illustrer l'itération d'un objet musical lors de la génération d'une partition ; on pourra lire un développement complet de cet exemple dans [56].



(a) Entrelacement total



(b) Avec une place de communication/synchronisation

FIG. 3.9 – Représentation d'un ordre partiel entre deux voix

Dans cet exemple, l'arc inhibiteur qui précède la transition finale empêche son franchissement avant 3 productions successives d'un jeton dans la place étiquetée $2B_1 + 2B_2$. Les termes B_1 et B_2 correspondent aux deux thèmes du *Boléro*, le majeur et le mineur. La production d'un jeton dans la place provoque l'ajout de la séquence $2B_1 + 2B_2$ à la partition générée ; à chaque production, le contenu musical de B_1 et B_2 étant modifié, les répétitions donnent lieu à des variations. Derrière cet exemple simple du pouvoir d'expression des réseaux, on entrevoit la possibilité de modifier le réseau pour avoir accès à d'autres partitions, ici en modifiant le nombre d'itérations grâce au poids de l'arc.

D'autres configurations s'attachent à reproduire des ordres partiels entre objets musicaux au travers de combinaisons de séquences et de synchronisations. Ils présentent l'exemple du jeu parallèle d'un soliste et de son accompagnement reproduit sur la figure 3.9.

Dans le premier cas, les deux parties sont purement parallèles et l'exécution du réseau peut produire n'importe quel entrelacement des différentes sections qui les composent. Dans la seconde configuration, des places booléennes de synchronisation intermédiaires contraignent plus fortement l'ordre partiel, permettant ainsi une communication entre les parties ; une analogie est faite entre ces places et les signes d'un chef d'orchestre. D'autres configurations sont envisagées pour l'indéterminisme ou la hiérarchie.

Une implémentation d'outils d'édition et d'exécution des réseaux réuni sous le nom de *MAP*[15] permet de générer réellement des partitions. Plusieurs versions sont possibles : la génération de fichiers sonores

[15] Musical Actors by Petri nets

représentant les différentes voix du morceaux synthétisées grâce au langage CMusic ([73]), la production d'événements MIDI ou de partitions symboliques dans le langage MCL . Les événements MIDI produits peuvent être envoyés directement vers un périphérique pour un jeu en temps réel de la partition. Dans ce contexte de "performance", l'utilisation de réseaux temporisés permet de laisser du temps s'écouler pendant le tir des transitons, modélisant ainsi le temps de jeu des événements MIDI par la place précédent la transition. Les auteurs font d'ailleurs remarquer que la prise en compte du temps dans l'exécution des réseaux n'est intéressante que dans ce cadre performance où le temps de la partition correspondant au temps du réseau, tandis que dans le cas de la génération d'une partition sous la forme d'un fichier, le temps d'exécution du réseau importe peu.

Transformations de réseaux

Un intérêt supplémentaire que Haus et al. voient dans les réseau de Petri est la possibilité de transformer un réseau d'origine représentant une famille de partitions en un autre réseau, et de générer ainsi des descriptions d'objets musicaux par modifications de descriptions existantes. L'abstraction des réseaux structurés leur permet de décrire ces transformations au moyen de réseaux de Petri de plus haut niveau. Les effets de bords provoqués par l'exécution d'un réseau de visent plus à générer une partition mais à modifier un réseau existant. Ces réseaux "matériau" étant des sous-réseaux du réseau de haut niveau, les auteurs s'appuient alors sur des propriétés d'auto-modification.

Les modifications de réseau qu'ils envisagent sont :
- modification de marquage initial
- modification de poids d'arc
- modification de structure
- transformation du contenu musical
- ...

Dans [24], Camurri et al. présentent plus précisément l'outil *MAP* et décrivent des transformations de structure de réseaux modifiant l'ordre partiel représenté par une configuration. La figure 3.7.1.0 illustre ces transformations.

Ces transformations modifient les ordres partiels entre les éléments musicaux d'une description par synchronisation et sérialisation de ces éléments.

Nous insistons quelque peu sur ces travaux car l'approche dont ils sont la cristallisation et certain de leurs développements, notamment les transformations sus-évoquées, présentent des points communs avec notre propre appréhension des réseaux de Petri. Dans la suite de ce manuscrit, la représentation des propriétés logiques et temporelles de partition est très précisément le rôle que nous donnons à cet outil, afin de disposer d'une machine d'exécution pour les partitions basée sur l'exécution des réseaux de Petri. Cependant des divergences apparaissent dans la génération et la manipulation des réseaux. Dans notre cas, l'utilisateur qu'il soit compositeur ou musicien n'est pas en contact direct avec les structures de réseau mais avec le langage de composition décrit dans la partie précédente. La création des réseaux ne se trouvent plus au centre du processus de composition ou d'analyse comme dans *MAP*, mais dans une étape transparente de traduction de partition interactive en réseau de Petri ; et si on retrouvera des transformations similaires à celles exposées ci-dessus c'est dans le cadre d'une compilation plutôt que dans une démarche de génération automatique par "dérivation" d'un matériau initial. Outre cette distance des réseaux à l'utilisateur, la seconde différence majeure entre ces travaux et les nôtres concernent la place du temps et de l'interaction. Pour le déroulement des partitions, nous nous plaçons systématiquement dans le cas où la temporalité du réseau est celle de la partition, ce temps d'exécution est donc particulièrement central pour nous. Bien que présente dans *MAP*, des questions de performance des machines de l'époque n'ont pas permis le développement de cette approche temps-réel.

A la suite de l'outil *MAP*, la poursuite des travaux basés sur les réseaux de Pétri ont conduit l'équipe du LIM à créer d'autres outils comme *ScoreSynth* [57] qui développe la représentation musicale par réseaux de Pétri, puis à la formalisation d'un système de plus haut niveau appelé *HARP* [23] qui bénéficie des apports de l'intelligence artificielle. Dans ce système, les relations logiques entre les éléments d'un matériau musical sont formalisées au travers de réseaux de "représentation de savoir" utilisant des outils IA comme les langages de la famille KL-ONE [21].

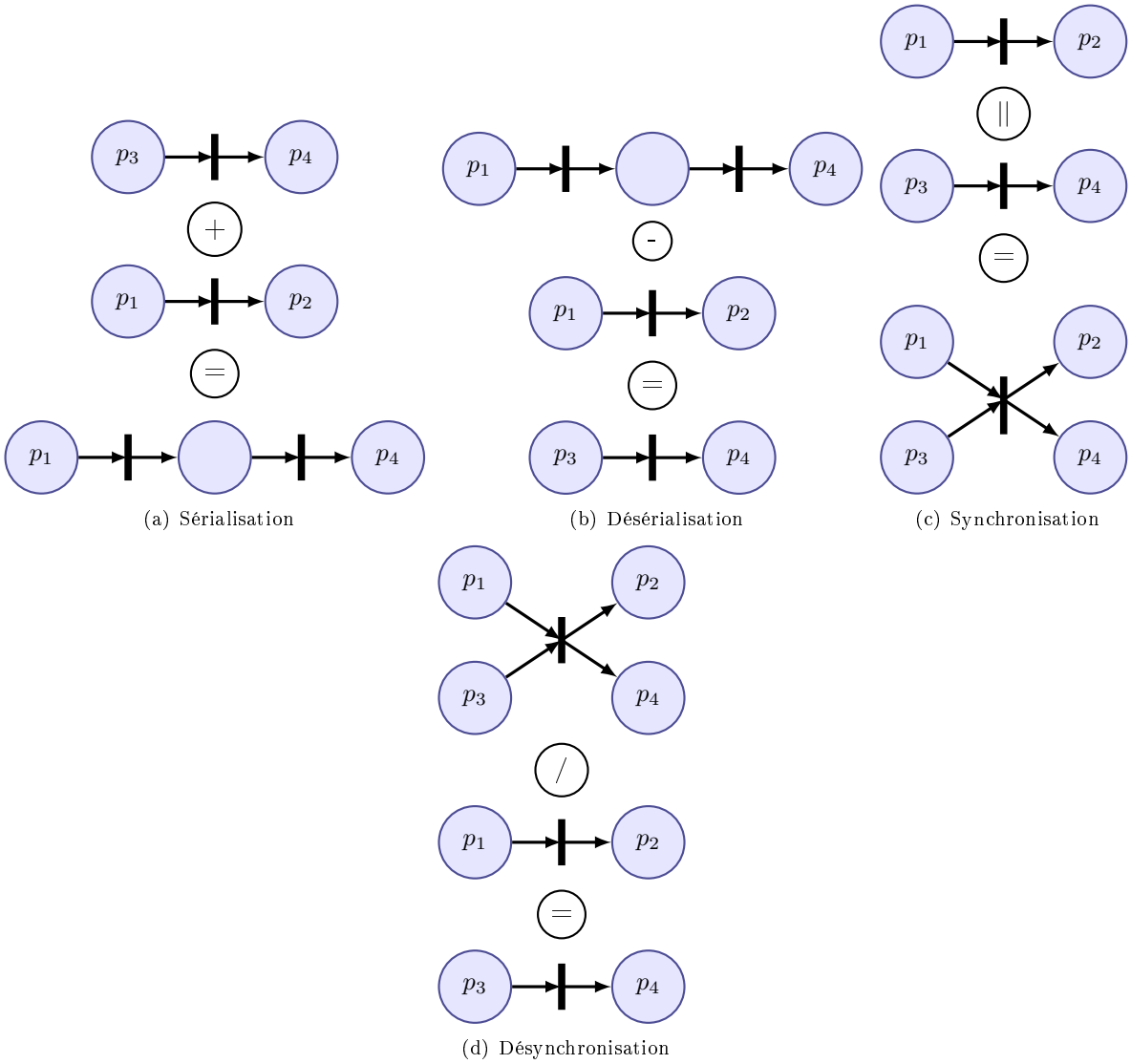


FIG. 3.10 – Transformations de structure de réseaux de Petri dans *MAP*

3.7.2 Autres utilisations

Les recherches menées par Haus et al. sont certainement celles qui ont poussé le plus loin l'utilisation des réseaux de Petri. Cependant pour être complet, il nous faut citer Stephen T. Pope qui à la même époque développa l'outil de génération automatique de partitions *DoubleTalk*[80]. L'utilisation des réseaux de Petri y est proche de celle de Haus et al. : réseaux de Petri hiérarchisés pour représenter un ensemble d'exécutions possibles d'une même partition. On peut toutefois noter l'utilisation de réseaux à prédicats, dont le franchissement des transitions est conditionné à la véracité d'une formule logique impliquant des descripteurs de l'état du réseau (temps écoulé) ou des caractéristiques de la partition.

3.7.3 Aujourd'hui

L'utilisation des réseaux de Petri a quelque peu décliné aujourd'hui pour la représentation musicale, bien que la recherche théorique les concernant reste vivace. Les limitations techniques des machines des année 80, interdisant de fait l'interaction temps réel avec l'exécution d'un réseau, mais aussi l'absence de modèle de réseaux temporels élaborés ont conduit à leur abandon au tournant des années 90. Cependant, l'évolution de ces modèles théoriques des réseaux temporels pourrait les remettre au goût du jour (et la suite de ce mémoire s'en veut la preuve).

Une autre approche de la représentation musicale pourrait remettre en lumière les réseaux de Petri. Nous faisons ici référence aux travaux de l'équipe *Représentation Musicale* de l'Ircam concernant l'analyse musicale assistée par ordinateur [9]. La démarche consiste ici à développer une *musicologie computationelle* à la suite de travaux mathématiques (Babbitt, Xénakis...), et de travaux informatiques comme ceux de Marcel Mesnage et André Riotte [71] autour de la modélisation de partitions. Il s'agit de s'appuyer sur des outils mathématiques (théorie des groupes) et d'informatique théorique pour analyser des pièces, en proposer des *modèles*, et éventuellement utiliser ces derniers pour composer de nouvelles pièces dans une logique d'analyse compositionnelle chère à Pierre Boulez [20]. Or certains travaux récents effectués dans ce cadre font apparaître une utilsation de structure de réseaux [3] laissant envisager une utilisation de réseaux de Petri.

Chapitre 4

Ordres partiels et réseaux d'occurrences

Comme nous l'avons évoqué dans le chapitre précédent, notre objectif est d'utiliser une représentation par réseau de Petri de la structure temporelle des partitions. Cette représentation doit comprendre à la fois des données quantitatives sur l'ordre des événements, ainsi que les caractéristiques quantitatives des intervalles séparant ces événements. En ce qui concerne la représentation d'un ordre partiel entre des événements, une structure spécifique de réseaux de Petri est particulièrement adaptée : les *réseaux d'occurrences.*

Dans notre cas, l'utilisateur, qu'il soit compositeur ou musicien, ne manipule pas des réseaux de Petri pour spécifier l'organisation de sa pièce. Il dispose d'un formalisme adapté à l'ecriture musicale que nous présentons dans la partie suivante. Par conséquent, la transformation d'une partiton en réseau de Petri en vue de son exécution, passe par une phase de compilation. Pour élaborer ce processus de transformation et nous assurer que le réseau d'occurences produit représente l'ordre temporel de la partition, nous nous appuyons sur les S-langages. D'abord nous exprimons les relations temporels contenues dans la partition sous forme de S-langage, puis nous construisons un réseau représentant ce S-langage.

Dans ce chapitre, nous faisons le lien entre réseaux d'occurences et S-langages. Nous commençons par rappeler quelques unes des propriétés des réseaux d'occurrences, puis donnons et démontrons les opérations permettant de passer d'une représentation à l'autre.

4.1 Définition et propriétés

Les réseaux d'occurrences ont été introduits par Nielsen dans [75]. Il s'agit de réseaux *place-transitions* dont la structure est quelque peu particulière.

Définition 4.1 *Un réseau de Petri marqué* $PN =< R, m_0 >$ *est un réseau d'occurrences si :*

- $\forall p \in P : |^\bullet p| \leq 1$
- $\forall p \in P : |^\bullet p| = 1 \Rightarrow m_0(p) = 0$
- $\forall p \in P : |^\bullet p| = 0 \Rightarrow m_0(p) = 1$
- $< R, m_0 >$ *est quasi-vivant*

Le marquage initial du réseau est déterminé par la structure de ce dernier. Il est d'usage de désigner le support de ce marquage, c'est à dire l'ensemble des places sans prédécesseurs, par $Min(R)$.

Nous rappelons à présent quelques propriétés immédiates des réseaux d'occurrence dont le lecteur pourra trouver les preuves dans [31].

Propriété 4.1 *Un réseau d'occurrences forme un graphe sans circuit et chaque nœud est précédé d'un nombre fini de nœuds.*

Une première conséquence de cette propriété est qu'il existe toujours dans un réseau d'occurences au moins une place sans prédécesseurs et au moins une place sans successeurs.

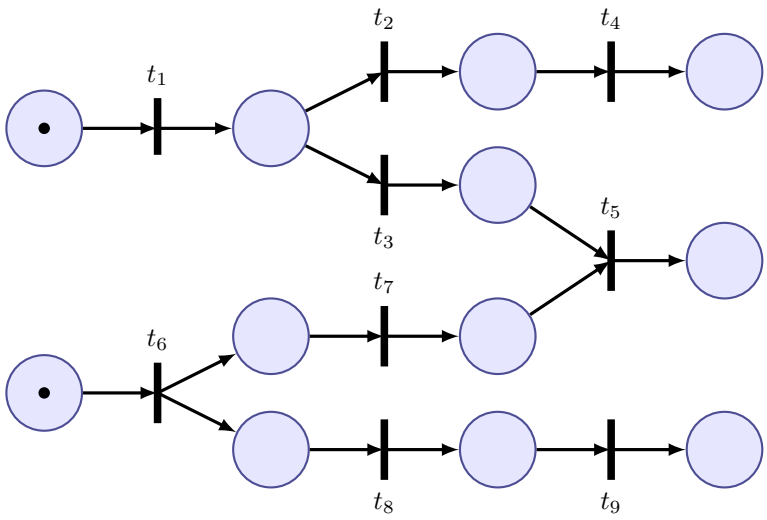


FIG. 4.1 – Un exemple de réseau d'occurrences

Propriété 4.2 *Un réseau d'occurrences est un réseau sain :*

$$\forall m \in A(R, m_0), \forall p \in P, m(p) \leq 1$$

De plus, les vecteurs caractéristiques des séquences sont sains i.e, une transition n'apparaît qu'une fois dans une séquence de franchissement.

Ces deux propriétés confèrent aux réseaux d'occurrence un statut privilégié pour représenter des ordres partiels entre événements. En associant un événement à une transition, on est assuré qu'au cours d'une séquence de franchissement un événement sera déclenché une seule fois.

De plus, la structure d'un réseau d'occurrences permet de comparer ses transitions deux à deux.

Définition 4.2 *Soit $PN =< R, m_0 >$ un réseau d'occurrences, avec P ses places, et T ses transitions, soient x et y deux nœuds de R $(x, y \in P \cup T)$:*

- x précède y *(que l'on notera $x \leq y$) si $x \in {}^*y$*
- x *et* y *sont* en conflit *(noté $x \sharp y$) si :*

$$\exists (t_x, t_y) \in T^2, t_x \neq t_y | t_x \leq x, t_y \leq y, {}^\bullet t_x \cap {}^\bullet t_y \neq \emptyset$$

- x *et* y *sont* concurrents *$(x||y)$ si ni $x \leq y$, ni $y \leq x$, ni $x \sharp y$*

Cette classification peut s'interpréter intuitivement : la précédence indique qu'une transition ne peut être franchie que si l'autre l'a été, le conflit traduit l'impossibilité d'avoir les deux transitions dans une même séquence de franchissement, enfin la concurrence indique que les transitions peuvent s'agencer dans n'importe quel ordre au sein des séquences de franchissements.

La figure 4.1 présente un exemple simple de réseau d'occurrences. Sur cet exemple, outre les relations de précédence immédiatement identifiables, on peut observer des transitions en conflit ($t_2 \sharp t_3$, $t_2 \sharp t_5$, $t_3 \sharp t_4$) et en concurrence ($t_1 || t_6$,$t_7 || t_8 \ldots$).

On peut voir dans un réseau d'occurrences, la modélisation de scénarios de déclenchements d'événements contenant des choix. En effet, si les notions de précédence et de concurrences permettent d'appréhender des ordres partiels entre des événements devant se dérouler dans une même exécution, la notion de conflit introduit l'exclusivité entre des choix de déroulements du scénario.

Pour caractériser formellement les déroulements possibles de l'exécution d'un réseau d'occurrence, est introduite la notion de *configuration*.

Définition 4.3 *Soit R un réseau d'occurrence,une configuration C de R est un sous-ensemble de T tel que :*

- $^\bullet C \subset C^\bullet \cup ||Min(R)||$
- $\forall t_1, t_2 \in C, \neg(t_1 \sharp t_2)$

Sur l'exemple de la figure 4.1, les ensembles $\{t_1, t_2, t_4\}$, $\{t_6, t_7, t_8\}$ ou encore $\{t_1, t_3, t_6, t_7, t_5, t_8, t_9\}$ sont des configurations du réseau.

Une configuration est un support d'une séquence franchissable. En ce sens, si l'on considère une exécution particulière du réseau d'occurrences, l'ensemble des transitions constituant un préfixe de cette exécution forme une configuration du réseau.

Cette dernière remarque est appuyée par deux propriétés importantes des configurations :

Propriété 4.3 *Soit R un réseau d'occurrences et C une de ses configurations, soit $\sigma = t_1 \dots tn$ une séquence de transition dont C est le support, alors σ est une séquence de franchissements depuis m_0 ssi :*

$$\forall i, j, t_i \leq t_j \Leftrightarrow i \leq j$$

Propriété 4.4 *Soit R un réseau d'occurrences et C une de ses configurations, soit $\sigma = t_1 \dots tn$ une séquence de franchissements à partir de m_0 dont C est le support. Soit $i \in [|1, n|]$:*

$$t_i || t_{i+1} \Rightarrow m_0[t_1 \dots t_{i+1} t_i \dots t_n >$$

Ces deux propriétés sont classiques pour des représentations d'ordres partiels, la première indique que si une transition en précède une autre, une séquence de franchissement impliquant les deux transitions les verra toujours apparaître dans le même ordre, tandis que la seconde propriété indique qu'on peut permuter des transitions concurrentes dans une séquence de franchissements.

4.2 Nouvelle sémantique de tir et S-langages

Cependant, cette dernière propriété fait apparaître un point d'achoppement concernant la représentation de la concurrence. En effet, la sémantique de tir classique dite de l'*entrelacement* oblige le franchissement d'une seule transition sensibilisée à la fois. Par conséquent, deux transitions concurrentes pourront être tirées l'une avant l'autre ou vice versa mais jamais simultanément. Cette constatation ne pose généralement pas de problèmes et dans le cadre d'analyse de systèmes modélisés par un réseau d'occurrences, on considère que deux transitions en concurrence représentent des événements pouvant s'agencer temporellement de n'importe quelle manière, et notamment qu'elles peuvent se produire simultanément.

4.2.1 Sémantique de tir "parallèle"

Pour répondre à ces deux objections, il nous faut changer de sémantique de tir et autoriser le franchissement simultané d'un sous-ensemble de transitions sensibilisées. Les implications d'une telle sémantique dans le cas général dépasse de loin la portée de ce manuscrit, nous nous l'adopterons donc uniquement pour les réseaux d'occurrences ; leur structure particulière et notamment la possibilité de classer les couples de transitions en fonction de leur "indépendance", nous permet de définir simplement la nouvelle sémantique de tir.

Définition 4.4 *Nous définissons une sémantique de tir dite "parallèle" pour les réseaux d'occurrences de la manière suivante :*

- *un sous-ensemble de transitions sensibilisées sont tirées*
- *deux transitions en conflit ne peuvent être tirées simultanément.*

On voit assez facilement que de par les caractéristiques des réseaux d'occurrences, cette sémantique de tir ne permet de tirer simultanément que des transitions en concurrence. En effet, deux transitions entretenant une relation de précédence ne peuvent pas être sensibilisées en même temps. Ceci correspond bien à la logique que nous cherchons à décrire.

Ce changement de sémantique implique cependant quelques modifications dans l'analyse des réseaux, et plus particulièrement dans l'analyse de l'accessibilité. On peut remarquer tout d'abord que la propriété 4.4 nous permet d'affirmer que l'ensemble des marquages accessibles d'un réseau d'occurrences reste le même malgré le changement de sémantique. Cependant, le graphe d'accessibilité voit l'ensemble de ses arcs augmenter, le tir simultané de plusieurs transitions depuis un marquage donné permettant de passer directement à des marquages accessibles par plusieurs tirs successifs avec la sémantique de l'entrelacement. Par conséquent la quasi-vivacité des réseaux d'occurrences reste valable avec la nouvelle sémantique.

4.2.2 S-langages de réseau

Un changement plus important intervient au niveau du langage de réseau. En effet, utiliser un alphabet classique pour les labels de transitions posent des problèmes car les mots du langage de réseau ne peuvent traduire la simultanéité de deux transitions. Pour pallier ce handicap d'expressivité, nous utilisons le formalisme des S-langages introduit au chapitre 3.

A partir de maintenant, on définit le S-langage d'un réseau d'occurrences de la manière suivante :

Définition 4.5 *Soit PN un réseau d'occurrences (son marquage initial m_0 est implicite), avec P et T ses ensembles de places et de transitions.*

En fixant M un ensemble de marquages terminaux, on peut définir le S-langage de PN associé à M $L(R, m_O, M)$ de la manière suivante :

$$L(R, m_0, Term) = \{u = u_1 \dots u_n \in \hat{T}^* | \exists m_f \in Term, m_0[u > m_f\}$$

L'alphabet utilisé pour $L(R, m_0, M)$ est T, de plus une S-lettre u_i de u représente le tir simultané de l'ensemble de transitions contenu dans u_i.

A partir de cette définition, on peut chercher à caractériser le S-langage d'un réseau d'occurrences par une expression algébrique de S-langage.

4.2.3 S-langage d'un réseau d'occurrences donné

Soient PN un réseau d'occurrences et L son S-langage défini pour un ensemble de marquages terminaux M.

Comme les séquences de franchissement dans PN sont saines par définition, alors une transition de T apparait au plus une fois dans un S-mot de L. En outre, l'existence éventuelle de situations de conflit implique que toutes les transitions de T n'apparaissent pas dans chaque S-mot de L.

On définit ainsi $\mathscr{U}_{|1|}$, le S-langage contenant tous les S-mots de $\hat{T}^*$ dans lesquels les transitions de T apparaissent au plus une fois :

$$\mathscr{U}_{|1|} = \bigcup_{\overrightarrow{p} \in \{0,1\}^n} \mathscr{U}(\hat{T}, \overrightarrow{p})$$

On a donc :

$$L \subseteq \mathscr{U}_{|1|}$$

Cependant, L ne contient pas systématiquement tous les S-mots de cette union, mais seulement ceux qui respectent la structure du réseau PN et qui s'arrêtent sur un marquage final. Nous commençons par exprimer le S-langage $\mathscr{L}(PN)$ qui représente la structure de PN indépendamment de l'ensemble M.

Nous introduisons quelques notations :

$$\begin{array}{rcl} e_k & = & (a_i)_{1 \leq i \leq n} \\ & & a_k = 1 \\ & & a_j = 0,\ j \neq k \end{array}$$

$$\overrightarrow{P_i} = \sum_{k \neq i} \overrightarrow{e_k} \text{ et } \mathscr{U}_i = \mathscr{U}(\hat{T}, \overrightarrow{P_i})$$

$\mathscr{U}_i$ est le sous-langage de $\mathscr{U}_{|1|}$ dans lequel la transition t_i n'apparait pas.

Pour calculer $\mathscr{L}(PN)$, il nous faut caractériser les relations entre transitions d'un réseau d'occurrences en utilisant le S-langage de ce réseau et l'opérateur de fusion.

Propriété 4.5 *Soient t_1 et t_2 deux transitions de PN, on a :*

- t_1 précède t_2 *ssi*
$$\mathscr{L}(PN) \subseteq \mathscr{U}_{|1|} \bowtie t_1 t_2$$
(t_1 et t_2 apparaissent toujours dans le même ordre)
- t_1 *est en* conflit *avec* t_2 *ssi*
$$\mathscr{L}(PN) \subseteq \mathscr{U}_{|1|} \bowtie (\mathscr{U}_i \cup \mathscr{U}_j)$$
(t_1 et t_2 ne peuvent pas apparaître dans le même S-mot)
- t_1 *est en* concurrence *avec* t_2 *ssi*
$$t_1 \otimes t_2 \subseteq \mathscr{L}(PN)$$
(toutes les configurations entre t_1 et t_2 sont possibles)

Pour calculer $\mathscr{L}(PN)$, il faut projeter $\mathscr{U}_{|1|}$ sur le S-langage associé à chaque couple de transitions de PN qui ne sont pas en concurrence.

Ansi :

$$\mathscr{L}(PN) = (\mathscr{U}_{|1|} \underset{t_i \leq t_j}{\bowtie} t_i t_j \underset{t_i \sharp t_j}{\bowtie} (\mathscr{U}_i \cup \mathscr{U}_j))$$

Pour obtenir la S-expression de L il faut restreindre $\mathscr{L}(PN)$ au S-mots conduisant de m_0 à un marquage final.

Soit un marquage m de M, on note L_m le sous S-langage de S-mots de $\mathscr{L}(PN)$ menant de m_0 à m. Pour exprimer L_m, il faut supprimer de $\mathscr{L}(PN)$ les transitions qui suivent les places du support de m (places telles que $m(p) \neq 0$). En notant P_m le support de m et

$$\alpha_m = T \setminus (\underset{p \in P_m}{\cup} p^* \cap T)$$

L'alphabet dans lequel on supprime les transitions suivant une place du support de m. On alors :

$$L_m \subseteq \mathscr{L}(PN)_{|\widehat{\alpha_m}}$$

Pour obtenir L_m, il ne faut conserver que les S-mots dont la dernière S-lettre contient une transition prédécesseur d'une place du support de m :

$$L_m = \{u = u_1 \dots u_n \in \mathscr{L}(PN)_{|\widehat{\alpha_m}} | \exists t \in \underset{p \in P_m}{\cup} {}^\bullet p, t \in u_n\}$$

Enfin,

$$L = \underset{m \in M}{\cup} L_m$$

Cette expression indique qu'un S-langage de réseau d'occurrences à une forme particulière exprimée par $\mathscr{L}(PN)$, à savoir la projection de $\mathscr{U}_{|1|}$ sur des S-langages traduisant la précédence et le conflit.

De plus on note qu'il est possible de choisir un ensemble de marquages finaux tel que :

$$L = \mathscr{L}(PN)$$

Il suffit de prendre M réduit à un seul marquage m dont le support est constitué des places de PN qui n'ont pas de successeurs. Le support de m n'est pas vide par définition des réseaux d'occurences et $\alpha_m = T$ d'où l'égalité ci-dessus.

Partant de ces constats, nous cherchons maintenant à construire un réseau d'occurrences à partir d'une S-expression donnée.

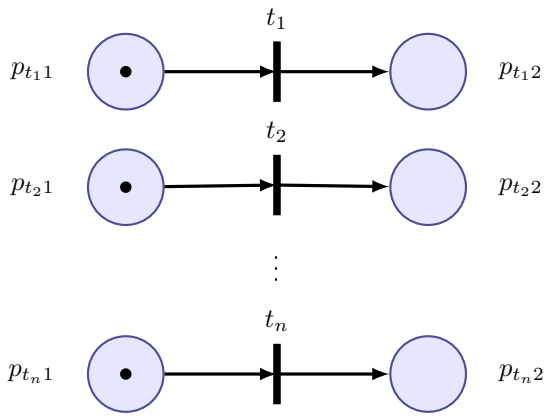


FIG. 4.2 – Un exemple de réseau univers pour un alphabet T

4.2.4 Construction d'un réseau à partir d'une S-expression

Soit une S-expression E définie sur un S-alphabet $\hat{T}$, de la forme d'un langage de réseau d'occurrences, c'est-à-dire :

$$E = (\mathscr{U}_{|1|} \underset{(i,j)\in P}{\bowtie} t_i t_j \underset{(k,l)\in S}{\bowtie} \{t_k, t_l\} \underset{(m,n)\in C}{\bowtie} (\mathscr{U}_m \cup \mathscr{U}_n))$$

avec $\mathscr{U}_{|1|}$ défini comme précédemment sur $\hat{T}$. Les ensembles P, S et C contiennent ici des couples d'indices permettant de désigner les S-mots pour la jointure avec $\mathscr{U}_{|1|}$.

Pour déterminer un réseau PN dont le S-langage est E, nous partons d'un réseau d'occurrences dont le langage $\mathscr{L}$ est $\mathscr{U}_{|1|}$, puis nous le modifions par transformations successives pour obtenir le S-langage recherché.

Réseau Univers

La construction du réseau est assez directe. Nous appelons ce réseau, le réseau univers pour l'alphabet T.

Définition 4.6 *Soit T un alphabet, le réseau* univers *de $T = \{t_1 \dots t_n\}$, noté R_T, est un réseau d'occurrences défini par :*

- $P = \bigcup_{t\in T}\{p_t1, p_t2\}$
- *l'ensemble des transitions de R_T est l'ensemble T lui même*
- Pre : $\forall t \in T, Pre(p_t1, t) = 1$
  Sinon $Pre(p, t) = 0$
- $Post$: $\forall x \in E, Post(p_t2, t) = 1$
  Sinon $Post(p, t) = 0$

Comme R_T est un réseau d'occurrences, son marquage initial est implicite. Il s'exprime comme suit :

$$m_0 = \{p_{t_i}1 | 1 \leq i \leq n\}$$

On peut trouver un exemple de réseau univers sur la figure 4.2

On peut vérifier que R_T est bien un réseau d'occurrences. De plus comme toutes les transition sont en concurrences, on a bien :

$$\mathscr{L}(R_T) = \mathscr{U}_{|1|}$$

Il nous faut à présent définir les transformations permettant de modifier le S-langage du réseau par jointure. Nous présentons également des résultats théoriques concernant ces transformations et les S-langages, pour le confort du lecteur nous présentons ici des démonstrations simplifiées basées sur la représentation graphique des réseaux de Petri. On trouvera des versions formelles de ces démonstrations dans l'annexe C. La figure 4.4 illustre ces opérations par des exemples graphiques, on pourra remarquer la proximité de celles-ci avec certaines tranformations utilisées par Haus et al. notamment leurs opérations de fusion et de sérialisation.

Opération de fusion

L'opération de fusion permet de modifier un réseau de telle manière que dans le S-langage du réseau obtenu, l'apparition deux S-lettres soit simultanée.

Définition 4.7 *Soit un réseau $PN =< R, m_0 >$ dont le S-langage est défini par $L = (R, m_0, l, Term)$. Soient t_i et t_j deux transitions de PN, l'opération de "*fusion*" de t_i et t_j consiste à remplacer t_i et t_j par une transition t_{ij} dans PN, conduisant au réseau $PN' =< R', m_0 >$ défini comme suit :*

- $P' = P$
- $T' = T \backslash \{t_1, t_2\} \cup t_{1.2}$
- Pre' : $\forall p \in P', Pre'(p, t_{1.2}) = Pre(p, t_1) + Pre(p, t_2)$
  $\forall (p, t) \in P' \times T' \backslash t_{1.2}, Pre'(p, t) = Pre(p, t)$
- *$Post'$ est défini de la même manière que Pre'*

Théorème 4.1 *Soit un réseau d'occurrences PN, soient t_1 et t_2 deux transitions* en concurrence *de PN, le réseau PN' issu de la fusion de t_1 et t_2 dans PN est un réseau d'occurrences et a pour S-langage de réseau tel que :*

$$\mathscr{L}(PN') = \mathscr{L}(PN) \bowtie \{t_1, t_2\}$$

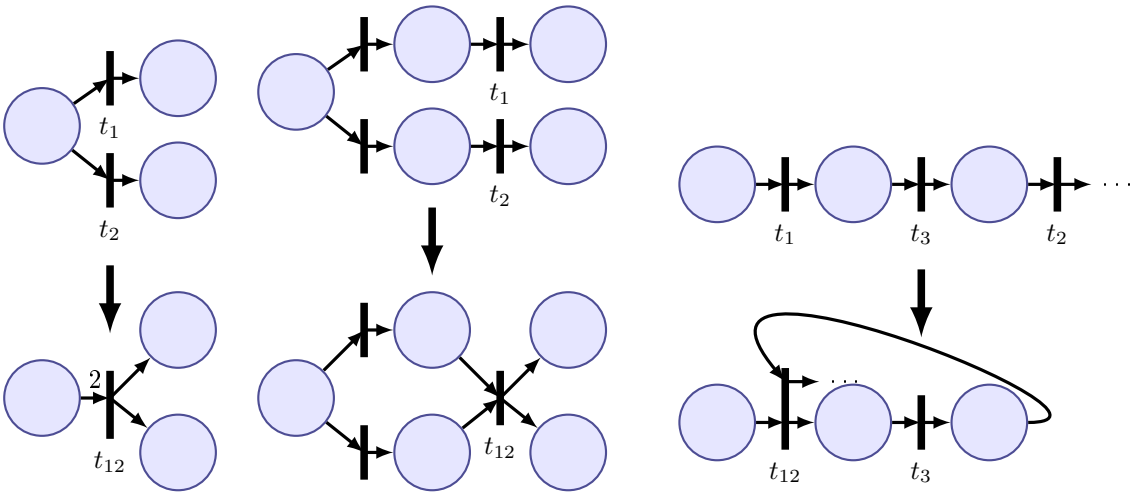
Remarque 4.1 *Dans le cas de deux transitions qui ne sont pas en concurrence, leur fusion conduit à un réseau qui n'est pas un réseau d'occurrence. Si les deux transitions entretiennent une relation de précédence, la fusion fait apparaître un circuit dans le réseau et si les transitions sont en conflit, la quasi-vivacité du réseau n'est plus assurée car aucun marquage ne peut sensibiliser la transition issue de la fusion. On notera le cas particulier de deux transitions partageant une même place "prédécesseur", qui conduit à l'apparition d'un arc de capacité 2, entre le prédécesseur commun et la nouvelle transition, interdisant la sensibilisation de cette dernière puisque le réseau est sain (le nombre de jeton dans une place est toujours inférieur à un). Des exemples graphique des ces situations sont présentés sur la figure 4.3. Ces résultats correspondent assez logiquement à des configurations temporelles faisant apparaître des incohérences temporelles.*

Mise en causalité

Comme son nom l'indique, il s'agit ici d'imposer un ordre entre deux S-lettres.

Définition 4.8 *Soient un réseau PN, t_1 et t_2 deux transitions de PN, l'opération de* mise en causalité *de t_i et t_j dans PN conduit au réseau PN' défini comme suit :*

- $P' = P \cup p_{12}$
- $T' = T$
- Pre' : $Pre'(p_{12}, t_2) = 1$
  $\forall t \in T \setminus t_2, Pre'(p_{12}, t) = 0$
  $\forall (p, t) \in P' \setminus p_{12} \times T', Pre'(p, t) = Pre(p, t)$
- $Post'$: $Post'(p_{12}, t_1) = 1$
  $\forall t \in T' \setminus t_1, Post'(p_{12}, t) = 0$
  $\forall (p, t) \in P' \setminus p_{12} \times T', Post'(p, t) = Post(p, t)$



(a) Apparition d'un arc de capacité 2 après fusion de deux transitions partageant une même place prédécesseur

(b) Apparition d'une transition non-sensibilisable après fusion de deux transitions en conflit

(c) Apparition d'un cycle après fusion de deux transitions entretenant une relation de précédence

FIG. 4.3 – Exemples graphiques d'incohérences temporelles liées à l'opération de fusion.

Théorème 4.2 *Soit PN un réseau d'occurrence, soient t_1 et t_2 deux transitions de PN telles que :*

$$t_1||t_2 \vee t_1 \leq t_2$$

le réseau PN' issu de la mise en causalité de t_1 et t_2 est un réseau d'occurrences tel que :

$$\mathscr{L}(PN') = \mathscr{L}(PN) \bowtie t_1 t_2$$

Mise en conflit

Il s'agit d'exclure les S-mots contenant en même temps deux S-lettres données.

Définition 4.9 *Soient un réseau PN, t_1 et t_2 deux transitions de PN, l'opération de mise en conflit de t_1 et t_2 dans PN conduit au réseau PN' défini comme suit :*

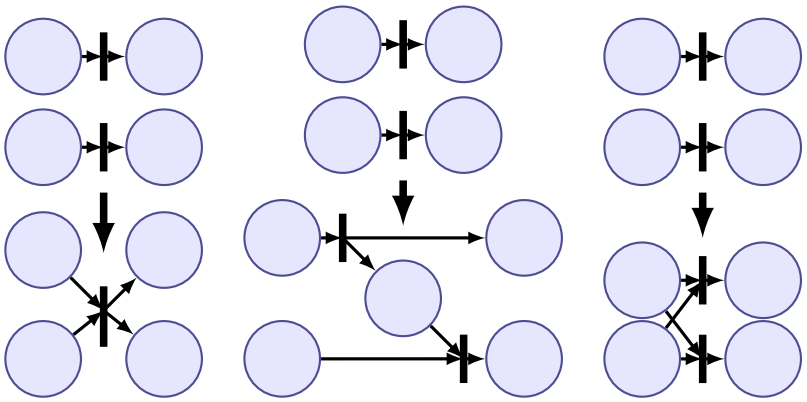
- $P' = P$
- $T' = T$
- Pre' : $\forall p \in P, (Pre(p,t_1) = 1) \vee (Pre(p,t_2) = 1) \Rightarrow (Pre'(p,t_1) = 1) \wedge (Pre'(p,t_2) = 1)$
  $\forall (p,t) \in P' \times T' \setminus t_1, t_2, Pre'(p,t) = Pre(p,t)$
- $\forall (p,t) \in P \times T, Post'(p,t) = Post(p,t)$

Théorème 4.3 *Soit PN un réseau d'occurrence, soient t_1 et t_2 deux transitions de PN telles :*

$$t_1||t_2 \vee t_1 \sharp t_2$$

le réseau PN' issu de la mise en conflit de t_1 et t_2 est un réseau d'occurrences tel que :

$$\mathscr{L}(PN') = \mathscr{L} \bowtie (\mathscr{U}_1 \vee \mathscr{U}_2)$$



(a) La fusion (b) Mise en causalité (c) Mise en conflit

FIG. 4.4 – Les transformations sur les réseaux d'occurrences

Construction

La construction du réseau d'occurrences dont le S-langage est donné par la S-expression E consiste donc à appliquer autant fois que nécessaire les transformations de fusion, causalité et conflit sur le réseau univers R_T. On peut noter que si E représente un S-langage vide, ce qui signifie qu'il existe une incohérence dans les relations entre transitions contenues dans E, alors le résultat des tranformations ne sera pas un réseau d'occurrences et son S-langage sera vide. Cette construction est donc valide même dans ce cas limite.

4.2.5 Places et S-langages

Jusqu'à présent nous nous sommes intéressés aux conséquences de la structure d'un réseau sur son S-langage, uniquement sous l'angle de ses transitions. On peut également caractériser certaines places en fonction de leur influence (ou de leur absence d'influence) sur le S-langage du réseau.

Places implicites

Un premier type de places caractérisables, sont les places dites *implicites* connues dans la théorie générales des réseaux de Petri. La formalisation des places implicites fait intervenir la notion de $p-flot$ dont la présentation n'aurait que peu d'intérêt ici. Intuitivement, une place implicite d'un réseau de Petri est une place qui ne peut empêcher à elle seule le franchissement d'une transition. La figure 4.5 présente un exemple classique de place implicite.

Un résultat connu et intéressant est que la suppression des places implicites dans un réseau de Petri préserve les propriétés du réseau, notamment celles concernant l'accessibilité.

En particulier, on observe aisément que la suppression d'une place ne modifie pas le S-langage d'un réseau d'occurrences. Dans le cas général, une place implicite peut avoir son importance si elle représente par exemple un état particulier du système modélisé par le réseau. Dans le ca squi nous intéresse ici, les places n'ont guère de sens propre, et la suppression de ces places ne pose donc aucun problème.

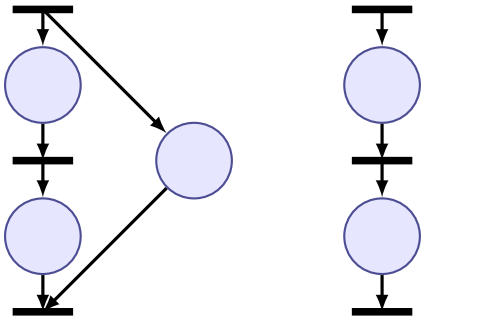
Se pencher sur le cas des places implicites est pertinent, dans la mesure où l'application des transformations successives sont susceptibles de faire apparaître des places implicites dans le réseau. Les deux lemmes qui suivent formalisent ce genre de situation.

Lemme 4.1 *Soient PN un réseau d'occurrences et t_1, t_2 deux transitions en concurrence dans PN telles que :*

$$^\bullet(^\bullet t_1) \cap {}^\bullet(^\bullet t_1) \neq \emptyset$$

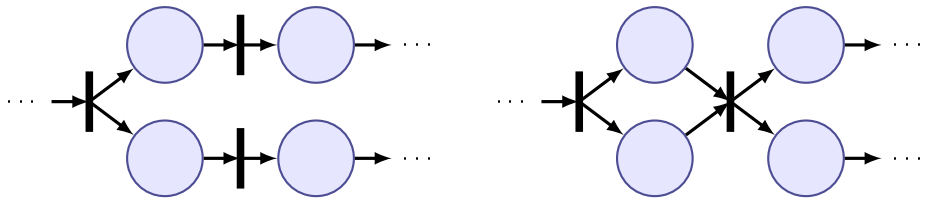
Dans le réseau PN' résultant de la fusion de t_1 et t_2 en une transition t_{12}, une place p de PN' telle que :

$$p^\bullet = \{t_{12}\}$$



(a) Une place implicite p (b) La suppression de la place implicite

FIG. 4.5 – Exemple de place implicite



(a) Situation initiale (b) Apparition de 2 places implicites après fusion

FIG. 4.6 – Places implicites après une fusion

est implicite.

Lemme 4.2 *Soient PN un réseau d'occurrences et t_1, t_2 deux transitions de PN telles que :*

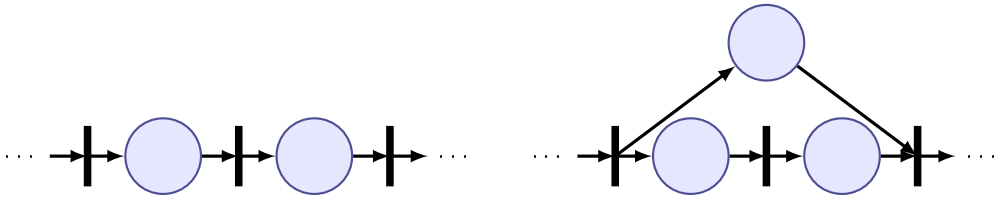
$$t_1 \leq t_2$$

Dans le réseau issu de la mise en causalité de t_1 et t_2, la place créée par l'opération est une place implicite.

On peut se convaincre de la véracité de ces lemmes par des exemples graphiques. Les figures 4.6 et 4.7 font office de preuve.

Places superflues

Dans le cas particuliers des réseaux d'occurrences, des places d'un autre type ne sont pas significatives pour le S-langage d'un réseau. Nous les appelons des places *superflues*. Encore une fois, dans le cas général



(a) Situation initiale (b) Ajout d'une place implicite après mise en causalité

FIG. 4.7 – Place implicite après mise en précédence

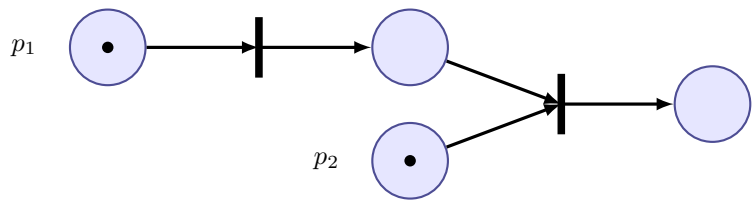


FIG. 4.8 – Un exemple de place superflue dans le support du marquage initial

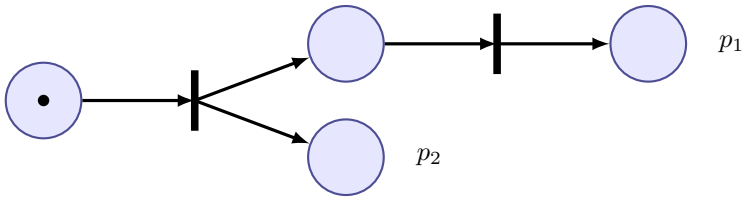


FIG. 4.9 – Un exemple de place sans successeurs superflus

elles peuvent être de première importance selon ce qu'elles modélisent mais ici leur suppression ne modifie pas le S-langage du réseaux d'occurrences.

On peut identifier deux types de places superflues, dans le support du marquage initial et parmi les places "finales", qui ne possèdent pas de successeurs.

Lemme 4.3 *Soient PN un réseau d'occurrences, soit une place p du support de m_0 telle que :*

$$p^\bullet = \{t\} \wedge {}^\bullet t \neq \{p\}$$

Alors p est superflue.

Lemme 4.4 *Soient PN un réseau d'occurrences, une place p de PN appartenant telle que :*

$$p^\bullet = \emptyset \wedge ({}^\bullet p)^*) \cap m \neq \{p\}$$

Alors p est superflue.

Les figures 4.8 et 4.9 présentent des exemples de places superflues. Comme pour les places implicites, leur suppression constitue une bonne simplification des réseaux.

La caractérisation des places implicites et superflues et leur suppression permettent de réduire la taille du réseau d'occurrences construit à partir d'une S-expression.

4.2.6 Discussion

Les résultats ici présentés nous sont utiles dans la suite pour représenter les partitions interactives dans une structure exécutable par la machine *ECO*. Nous utilisons donc la construction proposée pour compiler les partitions. On peut distinguer dès à présent notre emploi des réseaux de Petri de celui qu'en font Haus et al. ou Pope. En effet, dans leur cas le réseau est la structure de représentation musicale directement manipulée par l'utilisateur ; représentation qu'il peut transformer pour obtenir une autre partition ou plus exactement un autre ensemble de partitions. Dans notre cas, le compositeur manipule une partition écrite dans notre formalisme, partition dont les relations temporelles seront ensuite transcrites dans un réseau d'occurrences. L'écriture se fait au niveau de la partition non plus au nvieau du réseau. Naturellement il y a convergence au niveau des transformations, et on peut dire qu'il y a une analogie entre manipuler une relation temporelle dans notre modèle (ajout/suppression) et l'emploi des transformations dans *MAP*. Cependant, la création des réseaux dans notre modèle est algorithmique, là où elle est directement le fait de l'utilisateur chez Haus et Pope.

Deuxième partie

Vers un formalisme de partitions interactives

Résumé

Après avoir analysé les conditions et le cadre de l'interprétation musicale instrumentale, nous nous appuyons sur ce constat pour élaborer un formalisme général de partitions, qui inclut des éléments permettant "d'écrire l'interaction". Sous cette curieuse formule, nous désignons des objets capables de décrire les libertés laissées à l'interprète par le compositeur, ainsi que le cadre dans lequel celles-ci doivent s'inscrire. Ces outils sont pensés comme des contraintes entre différents paramètres de la partition. L'objectif est de permettre la spécification de partitions écrites, destinées à l'interprétation au sens instrumental, dans des contextes variés et notamment la musique électro-acoustique. Nous appelons ces partitions des "partitions interactives".

La présentation du formalisme couvre les trois chapitres de cette partie, l'un est consacré à la formalisation de l'organisation temporelle des partitions, indépendament de toute interaction. Le suivant expose les conséquences de cette caractérisation temporelle, au travers de liens implicites entre les objets d'une partition. Enfin le dernier introduit les éléments interactifs et les mécanismes les accompagnant.

Abstract

In the previous part, we analyse the possibilities for interpretation and the framework in which it can be expressed. Now we lay on this analysis to design a general formalism for musical scores, in which the interaction can be written. By "writing the interaction", we mean that the formalism contains some items that can describe the liberties given by the composer to the performer, and also the limit that the first one imposes to the second one. These items are though as some constraints defined between some parameters of the score. We aim to provide the composition of writen scores, that can be interpreted in an instrumental way, for various types of music, especially the electro-acoustic music. We call this type of scores "interactive scores".

We present the formalism in the three chapters of this part. The first one is about the description of the temporal organization of a score, without any type of interaction. The next one develops some consequences of the formalism for temporal organization, by exhibiting the implicit relations between the elements a score. The last chapter introduces the interactive items of the formalism.

Chapitre 5

Un nouveau type de partitions pour un modèle computationnel

L'approche de l'interprétation suppose la pré-existence d'un contenu musical qu'un interprète va venir manipuler par son jeu. C'est pourquoi nous distinguons dans la définition du formalisme *partitions statiques* et *partitions interactives*, Les premières représentent l'organistion musicale d'un matériau pensée par le compositeur, les secondes intègrent la possibilité pour un interprète de modifier cette organisation. Cependant, si les *partitions statiques* ne sont pas modifiables, l'organisation temporelles qu'elles contiennent s'exprime par des relations temporelles. Elles portent donc déjà les limites des libertés de modification qui seront définies dans les *partitions interactives*.

Nous présentons ici chacun des éléments présents dans les partitions statiques.

Remarque 5.1 *Il est important de noter pour la clarté des propos qui suivent que le terme de* partitions statiques *est à prendre dans le sens* partitions non interactives. *Il s'agit de distinguer ces deux types de partitions et non de considérer que les partitions statiques sont des partitions* rigides *et qu'elles ne peuvent donc être modifiées. Précisément certaines de leurs caractéristiques écrites peuvent se trouver modifiées lors de leur exécution. La distinction avec les partitions interactives se trouve dans les modalités de modification. Dans le cas interactif, les modifications sont le fait d'un interprète alors que dans le cas statique elles sont prdéfinies par le compositeur lui-même. Nous rappellerons cette remarque fondamentale régulièrement dans ce chapitre.*

Remarque 5.2 *Nous adoptons la conception du temps continu formalisée par Boulez dans l'opposition* temps strié/temps lisse *[19]. Dans cette conception, le temps lisse correpond au continuum absolu dans lequel s'incrivent les variations continues des paramètres physiques des processus sonores (amplitude de l'onde acoustique...). A l'inverse, le temps strié est celui de la structuration de partition (rythme, parties...). Le temps strié fait émerger des objets discrets dans le temps lisse grâce à des coupures dans le continuum, séparant celui-ci en "sous-continuums". L'apparition du discret dans le temps continu du rendu sonore permet la définition de structures et d'organisations, et donc la composition.*

Nous considérons ainsi les objets discrets manipulés dans les partitions ici définies, comme une structuration du continum temporel par des séparateurs. Par conséquent, les dates et durées dans les référentiels temporels prennent leur valeur dans $\mathbb{R}$.

5.1 Les objets temporels

Définition 5.1 *Un objet temporel OT est défini comme un 11-uplet :*

$$OT =< t, r, p, E, S, \mathscr{P}, \mathscr{V}, \mathscr{E}, \mathscr{B}, \mathscr{R}, \mathscr{C} >$$

dans lequel :

- t *est un type*
- r *est un rapport entre des quanta temporels*
- p *est un processus associé à OT*
- E *est un ensemble d'entrées*
- S *est un ensemble de sorties*
- $\mathscr{P}$ *est un ensemble ordonné de points de contrôle datés*
- $\mathscr{V}$ *est un ensemble de variables*
- $\mathscr{E}$ *est l'ensemble des objets temporels enfants de OT*
- $\mathscr{B}$ *est un ensemble de branchements*
- $\mathscr{R}$ *est un ensemble de relations i.e des contraintes locales*
- $\mathscr{C}$ *est un ensemble de contraintes globales*

Description

t **:** Il y a 2 possibilités pour le type t : *linéaire* ou *logique*, chacun correspond à un modèle temporel spécifique.

r **:** Il s'agit du rapport entre le quantum temporel de l'objet parent de OT et le quantum temporel de l'objet lui même.

p **:** Le processus p est l'opération réalisée par l'objet temporel pendant son déroulement. Nous abordons la notion de processus dans la section 5.2.

E **:** L'ensemble E contient des entrées permettant de fournir des valeurs en entrée au processus p.

S **:** L'ensemble S contient des sorties permettant de récupérer les valeurs de sortie du processus p.

$\mathscr{P}$ **:** Les points de contrôle de l'ensemble $\mathscr{P}$ sont des événements particularisés du déroulement de OT en lien avec des étapes du processus p.

$\mathscr{V}$ **:** L'ensemble $\mathscr{V}$ contient des variables dont les valeurs caractérisent l'état de l'objet. En ce sens, $\mathscr{V}$ constitue un environnement de OT. De nombreux paramètres peuvent être décrits par ces variables ; cette notion est approfondie dans la section 5.8.

$\mathscr{B}$ **:** L'ensemble $\mathscr{B}$ contient des branchements entre les entrées et sorties des objets enfants de OT. Ces branchements permettent des échanges de données produites par leurs processus. Ils peuvent également servir à communiquer des résultats de processus dans les niveaux hiérarchiques supérieurs de la partition ou encore vers l'environnement extérieur ; la section 5.7 apporte des précisions à leur sujet.

$\mathscr{R}$ **:** L'ensemble $\mathscr{R}$ contient des relations permettant l'organisation des objets enfants de OT. Il peut s'agir de relations temporelles ou logiques selon la nature de l'objet ; les sections 5.5 et 5.6 leurs sont consacrées.

$\mathscr{C}$ **:** L'ensemble $\mathscr{C}$ contient des contraintes impliquant les variables de l'ensemble $\mathscr{V}$. Ces contraintes peuvent être globales ou locales en se restreignant à un sous-ensemble de $\mathscr{V}$. Elles concernent aussi bien des caractéristiques temporelles que le contenu des objets temporels au travers des paramètres des processus associés aux objets et donc les valeurs admises par les entrées des objets temporels et produites sur leurs sorties.

Définition 5.2 *Si l'ensemble $\mathscr{E}$ d'un objet OT est vide alors OT est dit* **simple**, *dans le cas contraire il est dit* **complexe**.

5.1.1 Familles d'objets

Au dela du type t sur lequel nous nous pencherons plus en détails par la suite, nous pouvons distinguer principalement trois familles d'objets.

Définition 5.3 *Une* **texture** *est un objet simple représentant l'exécution d'un processus génératif, i.e. un processus qui produit des résultats liés aux entrées/sorties des partitions.*

Les textures sont ce que nous pourrions considérer comme les éléments de base des partitions. Un élan métaphorique avec la musique instrumentale pourrait nous pousser à les appeler des "notes". Cependant comme nous le verrons, une texture peut être associée à un processus dont les résultats ne sont pas directement diffusés au travers des sorties de la partition. Par conséquent cette appellation serait réductrice. L'exemple le plus simple d'une texture est certainement un objet représentant la lecture d'échantillons dans une table d'onde à une fréquence donnée. Les valeurs produites par une texture peuvent être redirigées directement vers la sortie de l'objet parent ou vers l'entrée d'un autre objet au moyen de *branchements*.

Définition 5.4 *De manière symétrique nous désignerons un objet simple associé à un processus non génératif par le terme* **objet contextuel**.

Le rôle des objets contextuels est de modifier le contexte musical de la partition. Ils peuvent agir essentiellement de deux manières : soit en modifiant l'environnement d'un objet complexe soit en générant automatiquement des objets intervenant à une date ultérieure de la leur au cours de la partition. Ils sont ainsi capables de modifier les contraintes de l'ensemble $\mathscr{C}$ ou encore la valeur des variables de l'ensemble $\mathscr{V}$ de leur objet parent. Les objets contextuels peuvent recouvrir des rôles très variés en fonction des variables et des contraintes qu'ils manipulent, à titre d'exemple, ils peuvent servir à limiter temporairement le nombre de processus exécutés simultanément, à modifier la dynamique des échantillons sonores produits par un objet ou encore changer une tonalité. La place qu'occupent ces objets apparaîtra plus clairement après les descriptions des variables d'environnement et des contraintes qui leurs sont attachées ; le lecteur trouvera en particulier dans la section 5.8 un exemple d'utilisation d'objets contextuels pour modifier un *tempo* associé à un objet et simuler un *accelerando*.

Définition 5.5 *A partir de maintenant, nous désignerons un objet complexe par le terme de* **structure**.

De façon relativement transparente, une structure correspond à l'organisation d'un sous-ensemble d'objets temporels, à leur structuration. Elle peut porter un ensemble de contraintes de contenu qui s'appliquent sur les processus de ses objets enfants. Une structure ne possède pas de processus associé quel qu'il soit. Les structures peuvent être vues comme des représentations d'algorithmes, ces algorithmes étant eux-mêmes assimilables à des processus particuliers. Pour préserver l'homogénéité des partitions, il n'est pas possible de mêler ces processus particuliers avec les processus des objets simples.

Remarque 5.3 *La disctinction faite entre les* **textures** *et les* **objets contextuels** *est une distinction entre les processus qui leurs sont associés. Dans un cas, le processus produit du contenu, dans l'autre il modifie l'environnement de l'objet parent, mais d'un point de vue strictement structurel il s'agit des mêmes objets.*

Les ensembles $\mathscr{R}$, $\mathscr{C}$, $\mathscr{B}$ et $\mathscr{E}$ concernent uniquement les structures et sont vides pour les textures et les objets contextuels. A l'inverse le processus p d'une structure est vide.

5.1.2 Convention graphique

Une figure facilitant grandement la compréhension d'un propos, nous utilisons par la suite nombre de représentations graphiques abstraites de partitions décrites avec notre formalisme. Arrêtons nous un instant sur les conventions de représentation des objets temporels et des différents éléments constitutifs des partitions. Nous présentons sur la figure 5.1 un exemple de partition.

Sur cette exemple, on trouve une structure S_1 contenant un objet OC ainsi qu'une autre structure S_2. Les éléments caractéristiques des structures ($\mathscr{V}$,$\mathscr{C}$ ...) sont figurés dans un cartouche situé en haut de chaque structure. Les points de contrôle des objets sont représentés par deux cercles placés en haut et en bas des objets, ces deux cercles ayant exactement le même sens. Les relations entre objets sont représentées par des flêches entre les points de contrôle, sur l'exemple on trouve une relation temporelle

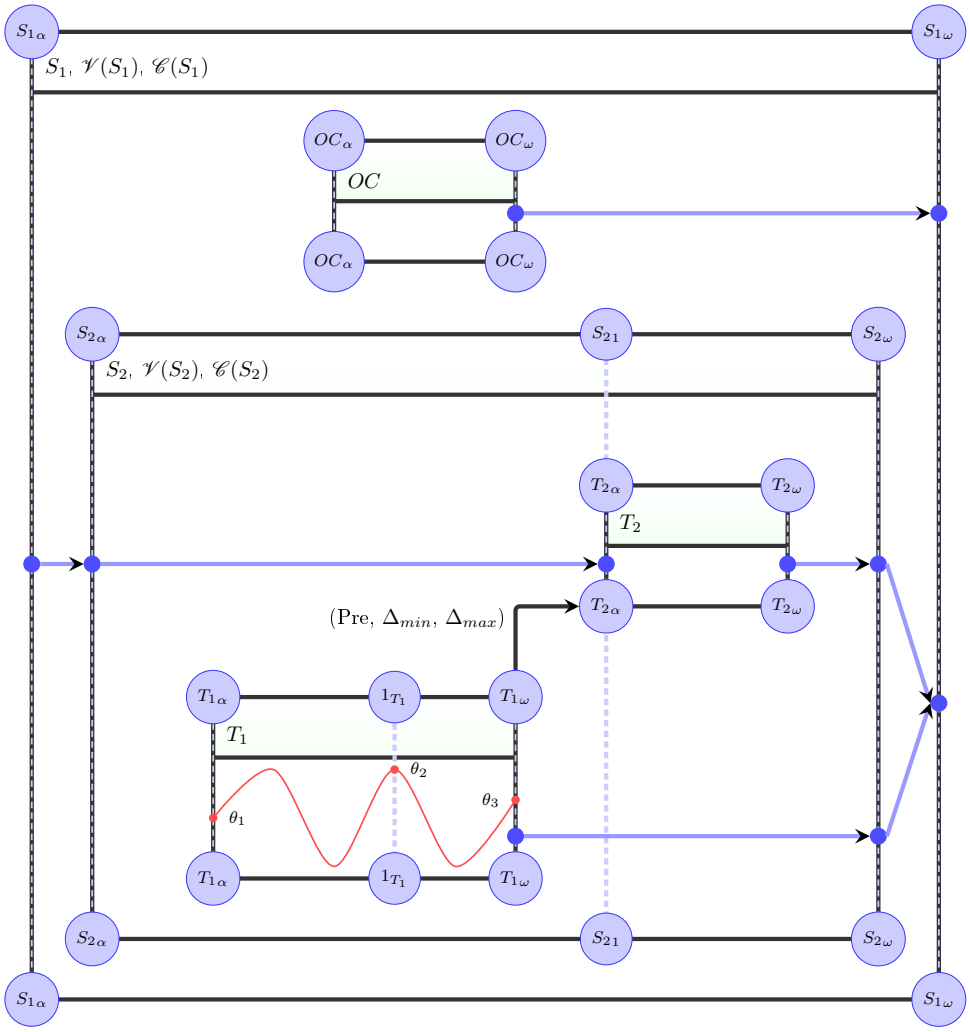


FIG. 5.1 – Un exemple de partition

entre T_1 et T_2. Les entrées et sorties sont les points placés à gauche et à droite des objets (à gauche pour les entrées, à droite pour les sorties). Les branchements sont des flêches reliant les entrées et sorties. Dans certains, nous représentons le processus associé à un objet comme par exemple T_1 qui effectue la lecture de la courbe représentée dans cet objet.

5.1.3 Quantum temporel

Avant de nous lancer dans la présentation des propriétés temporelles des structures, il est important de nous arrêter sur la notion de quantum temporel, commune aux deux types de structures. Le quantum d'une structure noté q est en fait une unité temporelle propre à la structure et dans laquelle vont s'exprimer ses caractéristiques temporelles ; on l'aura compris, c'est par rapport à cette unité que seront datés les événements de la structure. Naturellement, cette unité temporelle n'a de sens que par rapport à une unité de temps absolue servant de référence (la seconde par exemple). Ce quantum absolu permet de définir la valeur du quantum de la structure racine. Les valeurs des quanta des autres structures de la partition sont définies à partir du quantum de la racine en utilisant les rapports r introduits dans la définition des objets temporels. On peut calculer récursivement le quantum d'une structure à partir de celui de la structure racine grâce au rapport $r(S)$:

$$q(S) = r(S).q(parent(S))$$

Naturellement, cette formule ne s'applique pas à la structure racine celle-ci n'ayant pas de structure parent. En fait, le rapport $r(racine)$ permet de faire le lien entre le quantum de la structure racine, par extension le quantum de chaque structure, et une unité temporelle de référence. On a alors :

$$q(racine) = r_{racine}.q_{abs}$$

On peut voir le rapport r_{racine} comme un *tempo*. Notons que la valeur de ce rapport doit pouvoir être modifiée puisqu'une partition peut être exécutée à différentes vitesses. De la même manière, la valeur du quantum temporel d'une structure n'étant pas fixe, nous ne l'incluons pas dans sa définition, préférant insister sur les rapports qu'entretiennent les quanta des différents niveaux hiérarchiques. Notons cependant que le moment de l'éxecution d'une partition n'est pas le seul à faire appel au rapport $r(racine)$, mais la représentation graphique des partitions telle que nous l'envisageons implique également valuation des quanta des structures ; un rapport $r(racine)$ implicite permet alors de faire le lien avec l'échelle graphique utilisée pour représenter la référence q_{abs}.

Opérateur extension/compression

Un des avantages directs de la définition de quanta temporels hiérarchiques est la facilité avec laquelle ceux-ci permettent de définir certains opérateur courants. A titre d'exemple, nous présentons l'opérateur d'extension/compression temporelle que nous noterons classiquement *ext*. Cet opérateur prend en arguments une structure S et un rapport d'extension r_{ext} pour l'appliquer récursivement sur le sous-arbre dont S est la racine. La définition hiérachique des rapports de quanta simplifie à l'extrême l'application de cet opérateur puisqu'il consiste uniquement à modifier le rapport $r(S)$ en lui donnant la valeur $r(S).r_{ext}$. Ceci a pour conséquence directe de modifier le quantum temporel de la structure S et donc par définition ceux des structures du sous-arbre de S.

L'opérateur *ext* met en lumière l'intéret d'un système hiérarchique associé à l'utilisation des quanta temporels, pour doter le modèle des partitions d'une organistion multi-échelles bien adaptée à la représentation de la musique.

A présent, le moment est venu de présenter en détail le formalisme des partitions, nous nous appuierons pour ce faire sur leur structure hiérarchique par une description ascendante.

Nous entamons donc cette présentation par le matériau de base des partitions, les processus, c'est à dire le niveau *sub-symbolique* de notre représentation musicale.

Nous montrons par la suite comment organiser ce matériau dans une notation *symbolique* en établissant des liens avec des objets temporels, pour les organiser dans des structures de niveaux hiérarchiques de plus en plus élevés.

5.2 Les processus

Dans un sens, on peut voir les processus comme des calculs ou de la matière *hors-temps* dont les objets temporels sont des représentations *en-temps*. La distinction entre *hors-temps* et *en-temps* a été introduite par Xenakis ([97]). Selon lui, les éléments hors-temps sont générés indépendament de la temporalité de l'organisation dans laquelle ils seront impliqués. Le en-temps décrit précisément les liens temporels tissés entre les objets hors-temps au sein de structures temporelles. Nous adopterons cette approche, implicitement répandue parmi les outils de CAO.

Les objets temporels sont ainsi les représentations en-temps de processus hors-temps c'est à dire la représentation de l'exécution de ces processus dans le cadre d'une organisation plus large.

La distinction entre le hors-temps et le en-temps n'est cependant pas totalement hermétique. Ainsi, la "loi de composition interne" d'un objet hors-temps induit une temporalité intrinsèque régissant cet objet. C'est par exemple la féquence d'exécution d'un processus effectuant la lecture d'une courbe de valeurs ou la durée d'un processus consistant en la lecture d'un son enregistré. La mise en-temps des processus ne peut se faire totalement librement. C'est d'ailleurs dans ce dialogue entre le hors-temps et le en-temps que réside une part importante de l'écriture musicale.

Ainsi, les objets temporels permettent de plonger des processus possédant des temporalités intrinsèques dans une temporalité plus large (ou de plus haut niveau) correspondant à l'organisation temporelle de ces processus entre eux pour former le discours musical de la pièce.

Il est clair que les propriétés temporelles d'un objet représentant l'exécution d'un processus particulier sont liées à la temporalité intrinsèque de ce processus. Le compositeur ne pourra jamais totalement s'affranchir des caractéristiques temporelles des processus. Cependant, certains d'entre eux sont suceptibles d'accepter des modifications de leur temporalité, comme par exemple la durée qui sépare deux moments spécifiques de leur exécution. Il est par conséquent nécessaire qu'au moment de la composition (de la mise en temps des processus), les caractéristiques temporelles des processus soient identifiées, et que les modifications possibles de ces caractéristiques soient connues. Dans une première analyse, il apparait que la caractéristique temporelle non modifiable partagée par l'ensemble des processus est que leur fin ne peut jamais précéder leur début. Nous verrons comment généraliser cette approche pour des processus possédant des étapes intermédiaires et également que cette assertion constitue une des limites de notre modèle.

Définition 5.6 *Un processus* p *est un 7-uplet :*

$$p =< t, E, S, C, \Theta, \Gamma_t, \Gamma_{nt} >$$

où :

- t *est un type*
- E *sont des entrées*
- S *sont des sorties*
- C *est un calcul*
- Θ *est un ensemble ordonné d'étapes du calcul* C
- Γ_t *est un ensemble de contraintes temporelles relatives à la temporalité intrinsèque du processus*
- Γ_{nt} *est un ensemble de contraintes non-temporelles portant sur les paramêtres du processus*

Description

***t* :** Le type t constitue une première tentative de classification des processus. Nous avons distingué deux familles de processus : les processus *génératifs* concernent des opérations directement liées aux données produites par la partition pendant son exécution, les processus *non génératifs* concernant quant à eux des opérations liées à la stucture interne de la partition. Le type du processus permet d'identifier l'opération réalisée par le processus et éventuellement en déduire quelques caractéristiques intrinsèques. Ces types ne sont pas indépendants, et sont liés à l'approche qu'en a l'utilisateur. Il y a 5 types dit *génératifs* :

- *synthèse* : un processus de synthèse possède au moins une sortie.
- *traitement* : un processus de traitement possède au moins une entrée et une sortie.
- *diffusion* : un processus de diffusion possède au moins une sortie et se distingue des processus de synthèse en ce qu'il n'effectue par d'opérations de calcul pour produire ses résultats mais renvoie des données pré-existentes (lecture d'un fichier ou diffusion de données préalablement produites par un autre processus).
- *acquisition* : un processus d'acquisition possède au moins une entrée.
- *état* : les processus de ce type correspondent à la mise dans un certain état d'un matériel ou d'une application logicielle (chargement de presets ou mise sous tension d'un appareil). Ces processus possèdent nécessairement une contrainte intrinsèque sur leur durée traduisant le temps nécessaire au déroulement de l'opération extérieure à la partition qu'ils représentent.

et 2 type *non-génératifs* :

- *contrainte* : un processus de contrainte modifie les environnements des structures
- *structurel* : un processus structurel produit non pas des résultats liés aux sorties de la partition, mais des objets temporels et éventuellement une organisation associée qui seront utilisés plus tard dans le déroulement de la partition. Il permet donc de modifier la structure interne de la partition. Il peut s'agir par exemple d'une analyse "haut niveau" du jeu d'un musicien, produisant un objet temporel pour chaque note jouée organisés selon le jeu du musicien.

Rappelons les liens entre les types de processus et les familles d'objets qui les représentent, les *textures* portent des processus génératifs, les *objets contextuels* portent des processus *non génératifs*, les structures n'étant associées à aucun processus.

Quels que soient les raffinements futurs autour de ce typage et de ses sous-typages éventuels, le type *état* devra toujours apparaître pour permettre d'identifier des processus en lien avec des changements d'état de matériels physiques.

E **:** Les entrées et sorties E et S permettent au processus de récupérer des données pour effectuer son calcul et de diffuser ses résultats. Une classification des processus peut également s'opérer sur le nombre de leurs entrées et sorties.

C **:** Le calcul C est l'opération algorithmique réalisée par le procesus. Celui-ci s'exécute séquentiellement et est susceptible de produire des résultats sur ses sorties tout au long de son déroulement et non pas uniquement lorsqu'il arrive à son terme.

Θ **:** L'ensemble $\Theta = \{\theta_1, \ldots \theta_i \ldots, \theta_m\}$ est une liste finie chronologiquement triée d'étapes précises du calcul C. Il ne s'agit pas nécessairement des moments de production de résultats par C. Les étapes les plus naturelles d'un processus sont son début et sa fin.

Γ_t **:** Les contraintes temporelles Γ_t sont des contraintes quantitatives ou logiques entre les étapes θ_i. Il peut donc s'agir de contraintes sur les intervalles de temps qui séparent les étapes du processus, ou encore des relations logiques entre ces étapes. Par exemple, un processus de traitement réalisant l'inversion temporelle d'un extrait audio capté en direct, ne pourra pas commencer son calcul avant l'acquisition du dernier échantillon. Il y aura donc une contrainte de précédence entre la fin du processus d'acquisition des échantillons audio fournissant les données au processus de traitement et le début de ce processus de traitement. Un autre exemple de contraintes temporelles pourrait être la fréquence d'échantillonnage d'un flux produit en sortie. Cette ensemble contient donc les manifestations de la temporalité intrinsèque du processus.

Γ_{nt} **:** Les contraintes non temporelles Γ_{nt} sont des contraintes sur les paramètres du calcul C à savoir les valeurs qu'il accepte en entrée et celles qu'il produit en sortie. Par exemple des typages ou des intervalles de valeurs.

Revenons un instant sur les étapes de calcul des processus. Notons tout d'abord, que la liste des étapes de calcul est telle qu'à chaque exécution du processus, toutes les étapes se produisent et toujours dans l'ordre chronologique indiqué. En outre à la différence des objets temporels, il n'existe pas de processus ponctuel, par conséquent un processus à toujours un début et une fin. Eventuellement, la fin du processus est inaccessible dans le cas d'un calcul faisant apparaître une boucle infinie, comme nous le verrons plus loin, les point de contrôle sont alors fort utiles.

Insistons également sur le fait que les processus produisent des résultats non pas uniquement au terme de leur exécution mais éventuellement tout au long de celle-ci. Un processus produit dans ce cas un **flux** de valeurs.

Pour illustrer cette notion d'étapes de calcul, on peut prendre l'exemple d'un processus consistant en la lecture d'une courbe de valeurs à une fréquence d'échantillonage de 44100 Hz. Il ne semble pas nécessairement pertinent que chaque moment de production d'une valeur soit une étape caractérisée du processus. Cependant, il parait plus significatif que les étapes du calcul correspondent par exemple à des points d'inflexion de la courbe de valeurs.

Discussion

Il convient d'effectuer plusieurs remarques concernant la possibilité de représenter n'importe quel calcul impliqué dans une partition de la manière dont nous les avons décrits. Tout d'abord, la caractérisation des processus par une suite ordonnée d'étapes de calcul se réalisant toutes à chaque exécution du processus et surtout dans le même ordre limite de facto les types de processus caractérisables. Les processus représentables de cette manière sont ceux dont le calcul associé est un algorithme déterministe non conditionnel, c'est à dire clairement une suite d'opérations séquencielles. Des processus mettant en œuvre des algorithmes impliquant des choix, qu'ils soient déterministes ou probabilistes ne peuvent exhiber des étapes ordonnées se produisant à chaque exécution. Il n'est donc pas possible de représenter l'exécution d'un tel algorithme avec un objet simple. Nous verrons qu'il est cependant possible de représenter ces algorithmes à l'aide de structures en les décomposant en parties "atomiques", en associant ces parties à des objets simples et en organisant ces objets avec des relations.

On peut toutefois noter que le nombre d'étapes exhibées pour le calcul d'un processus dépend du degré de précision que l'on se fixe dans sa description formelle. Entre les deux niveaux extrèmes d'accuité que constituent d'une part la représentation de chaque opération algorithmique, et la représentation simplement du début et de la fin du processus d'autre part, un large choix de représentations d'étapes de calcul est possible. Ainsi, il est tout à fait envisageable de ne pas représenter des étapes de calcul qui invalideraient les propriétés de l'ensemble Θ concernant le nombre et l'ordre constants des étapes de calcul. Par exemple, ne pas représenter des étapes incluses dans une structure conditionelle évite les désagréments cités plus haut. C'est déjà ce que nous sous-entendions au sujet d'algorithmes contenant une boucle infinie, associé à un processus disposant d'un nombre fini d'étapes de calcul. Cette association n'est possible que si les étapes de la boucle infinie ne sont pas représentées. Naturellement, un nombre moindre d'étapes de calcul représentées implique une finesse limitée dans la description des contraintes intrinsèques du processus, car certaines d'entre elles sont susceptibles d'impliquer des étapes non représentées. Dans l'idéal, il serait souhaitable que notre formalisme permette la représentation de tous types de processus (en utilisant les structures pour le cas complexe) sans recourir à ce type d'artifices.

En outre, la notion de processus pose la question d'un formalisme permettant la description des calculs qui leur sont associés. En effet, les possibilités envisagées de déduire de la description d'un calcul des étapes de ce calcul ainsi que des propriétés logiques et temporelles nécessitent une représentation formelle manipulable de ces calculs. Naturellement l'informatique théorique regorge de travaux et de puissants résultats en algorithmique, mais le choix d'un type de représentation dans la littérature est déterminé par la connaissance des algorithmes les plus souvent impliqués. Un travail de ce type a été entamé dans le cadre du langage Faust [**?**] dont le but est justement de formaliser les algorithmes impliqués dans les applications de traitement de signal pour la musique. Il pourrait s'avérer intéressant de lier le modèle de partitions interactives avec le langage Faust en s'appuyant sur ce dernier pour représenter les calculs et en tirer les propriétés intrinsèques des processus dont nous avons besoin, cette éventualité reste à explorer.

Après cet aperçu de l'argile avec lauqelle on façonne les partitions, il est temps de voir comment les transformer en briques pour les assembler en des architectures temporelles élaborées. Comme annoncé, la mise en temps des processus se fait par le biais des objets temporels simples qui constituent la base de la notation symbolique des partitions interactives.

5.3 Objets simples et points de contrôle

Un objet simple est la représentation d'une exécution du processus qui lui est associé, il paraît donc essentiel de permettre de représenter au niveau de l'objet temporel le déroulement des étapes de calcul successives de son processus. C'est le rôle des points de contrôle.

On peut donc faire correspondre des étapes de calcul d'un processus avec les points de contrôle de l'objet temporel auquel il est associé.

Définition 5.7 *Soit un objet temporel* OT *et son ensemble de points de contrôle* $\mathscr{P}$*, soit le processus* p *associé à* OT *et* Θ *son ensemble d'étapes de calcul, on définit la fonction control :*

$$control : \mathscr{P} \rightarrow \Theta$$

telle que control(pc) est l'étape du calcul de p déclenchée par pc.

Les points de contrôle et la fonction *control* sont ainsi les liens qui articulent les niveaux *sub-symbolique* et *symbolique* de notre notation.

Assez logiquement, le début de OT contrôle le début de p et si elle existe, la fin de OT contrôle la fin de p.

Chaque point de contrôle de OT contrôle le déclenchement d'une unique étape du calcul de p, et il est important de noter qu'un point de contrôle de OT n'a pas de raison d'exister s'il ne contrôle pas une étape de p. A l'inverse, toutes les étapes de p ne sont pas nécessairement contrôlées par un point de contrôle de OT. La fonction *control* est une injection de $\mathscr{P}$ dans Θ.

On a donc :

Propriété 5.1

$$card(\mathscr{P}(OT)) \leq card(\Theta(p))$$

On peut mettre en relation cette propriété avec la remarque précédente concernant la précision avec laquelle on décrit les processus. L'objet temporel constitue une abstraction supplémentaire du calcul algorithmique effectué par son processus, sa représentation sous la forme d'une suite d'étapes séquentielles en étant une première. De la même manière qu'au moment du choix des étapes de calcul, nous avions envisagé la possibilité de jeter un voile pudique sur des caractéristiques embarassantes d'un algorithme, il est possible lors de la mise en temps d'un processus de ne conserver que certaines étapes représentées. Bien évidemment, un plus grand nombre d'étapes représentées implique la possibilité d'une d'organisation temporelle plus fine. Cependant, un objet simple devient ici une sorte de boite noire à qui on peut fournir des valeurs en entrée, en récupérer en sortie et dont on ne connait que les propriétés sucseptibles de nous intéresser. Un objet temporel peut ainsi être vu comme une "trace" d'exécution d'un processus, indépendante de l'opération concrète réalisée par ce processus.

La différence majeure entre les points de contrôle et les étapes de calcul est que les points de contrôle sont datés alors que les étapes ne le sont pas. C'est donc en ce sens qu'un objet temporel permet de plonger un calcul "hors-temps" dans la temporalité de la partition : en datant les étapes de p par l'intermédiaire des point de contrôle.

Laissons de coté pour l'instant l'épineuse question de la ligne de temps dans laquelle les dates des points de contrôle s'expriment. Considérons pour l'instant que chaque objet simple OT dispose d'un référentiel temporel dont l'origine coïncide avec son début, et que les points de contrôle de OT prennent leur dates dans ce référentiel.

Précisons que les points de contrôle naturels d'un objet sont son début et sa fin. Dans le cas d'un objet ponctuel, son unique événement est son unique point de contrôle.

La représentation des dates des points de contrôles s'opère dans le référentiel de l'objet temporel. Nous utiliserons donc pour la notion d'*oblitération temporelle* (*time-stamp*) formalisée par Mira Balaban et qu'on retrouve dans les modèles hiérarchiques. Nous utiliserons ici la notion courante @.

Formulation 5.1 *En notant* $start(OT)$ *et* $end(OT)$ *les points de contrôle associés au début et à la fin de OT on a :*

- *Si OT est ponctuel alors* $\mathscr{P} = \{pc_0@d_0\}$
- *Sinon* $\mathscr{P} = \{pc_0@d_0, \ldots, pc_i@d_i, \ldots, pc_n@d_n\}$

Avec :

- $pc_0 = start(OT)$ *et* $d_0 = 0$
- $pc_n = end(OT)$

La temporalité intrinsèque du processus peut influencer sa mise en temps au travers des points de contrôle. On peut considérer que cette temporalité va limiter les dates possibles pour les points de contrôle, qu'elle va imposer des **contraintes** sur ces dates, précisément des contraintes contenues dans l'ensemble Γ_t de p. Ces contraintes étant liées à la loi de composition interne du processus, celles-ci ne pourront pas être violées dans le cadre de l'organisation de la partition. Ce sont des règles immuables auxquelles le compositeur doit se plier.

En outre, cette mise en temps des processus n'est pas sans nécessiter des adaptations de leur part. Il faut que leur calcul soit capable d'accepter, dans le respect de ses contraintes intrinsèques, différentes dates pour ses étapes de calcul.

Certaines situations peuvent en effet s'avérer délicates. Pour l'illustrer, nous présentons sur la figure 5.3 l'exemple de la mise en temps d'un processus ayant la contrainte de produire des résultats à une fréquence donnée en lisant dans une table de valeurs (synthèse par lecture de table d'onde par exemple).

Dans la première situation 5.2(a), la mise en temps permet une lecture normale des valeurs, c'est à dire que les dates des points de contrôles imposées aux étapes de calcul correspondent au nombre de valeurs dans la table et sa fréquence de lecture. Dans la deuxième situation, on suppose que le point de contrôle final est placé à une date postérieure à la situation précédente. Cette mise en temps nécessite de déterminer de comportement du processus pendant la durée qui sépare d_2 et d'_2. Une très large variété de stratégies sont possibles, sur notre exemple nous avons supposé que pour combler cette durée, le processus prolongeait la courbe de valeurs en suivant une tangente. Nous aurions très bien pu considérer que le processus rééchantillonerait ses valeurs d'entrée ou encore ne produirait aucune valeur.

La situation inverse pourrait également se produire avec une date de point de contrôle antérieure à une situation normale. On peut faire ici le lien avec l'exemple d'un processus comprenant une boucle infinie. Dans ce cas, la fin du processus ne peut être datée (ou alors avec une valeur infinie) et le choix d'une date du point de contrôle de fin provoquera nécessairement l'anticipation de son étape finale. Il est donc important que le processus soit capable de s'adapter à cette anticipation en renvoyant par exemple une valeur de résultat intermédiaire qu'il aura atteinte au moment du déclenchement forcé de sa fin. Remarquons au passage qu'il est nécessaire que la terminaison des calculs des processus associés aux objets temporels ponctuels soit assurée. Ces objets ne disposant pas de point de contrôle final, une non terminaison du processus rendrait impossible leur arrêt.

5.3.1 Les événements

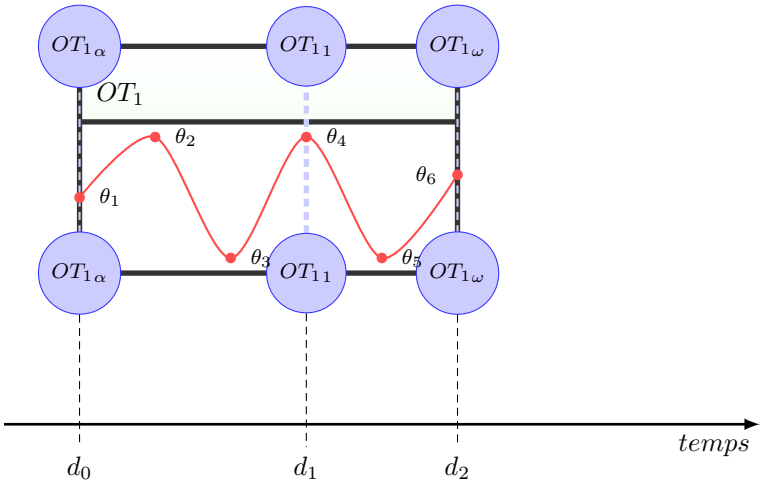
Les points de contrôle correspondent à des moments importants d'une partition, les seuls que l'on peut caractériser explicitement parmi le déroulement d'une partition. Plus précisément et pour reprendre une métaphore déjà usitée, les points de contrôle datés forment une véritable trace du déroulement de la partition. Dans un parallèle avec les conceptions philosophiques du temps, nous introduisons la notion d'*événement*.

Nous définissons les événements hérarchiquement :

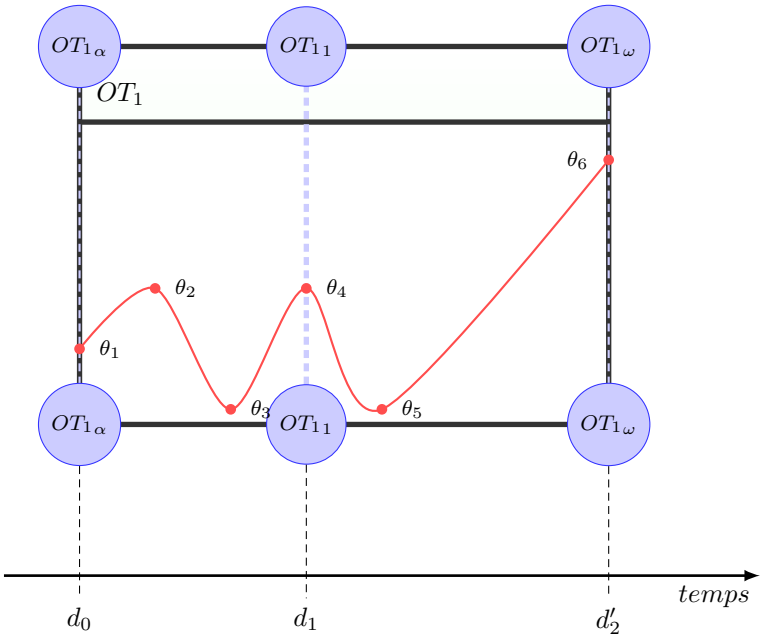
Définition 5.8 *Soit St une structure en notant* $\Sigma(St)$ *l'ensemble des événements de St on a :*

$$\Sigma(St) = \{start(St),\ end(St)\} \cup \{ \bigcup_{OT \in \mathscr{E}(St)} \mathscr{P}(OT)\}$$

On peut voir les événements comme l'ensemble des moments caractérisables d'une structure (d'un niveau hiérarchique) de la même manière que les étapes de calcul sont des moments identifés d'un processus. Par extension, il est possible d'assimiler les événements d'une structure aux étapes d'un "processus"



(a) Une mise en temps d'un processus



(b) Une autre mise en temps du même processus

FIG. 5.2 – Un exemple d'association d'un processus à un objet temporel. La mise en temps du processus au travers de l'objet peut conduire à la troncature ou l'augmentation de la courbe. Les modalités de la création de valeurs supplémentaires doivent être définie dans la formalisation de du processus.

particulier porté par elle. La notion d'événement est d'ailleurs centrale dans l'organisation temporelle des objets au sein de structures.

Nous considérons que les événements d'une partition sont l'ensemble des événements de ses structures.

5.4 Les structures

Les différents modèles temporels pour la composition musicale que nous avons identifiés précédemment font apparaître un point d'achoppement important autour de la représentation des instances d'un objet temporel. Les modèles hiérachiques avec relations temporelles se montrent en effet incapables d'associer à la même représentation d'un objet plusieurs instances de cet objet ; ce faisant, ils perdraient la notion d'orientation du temps et rendraient inopérants les raisonnements sur les relations temporelles. Cependant, les représentations basées sur des modèles de type états/transitions sont suffisamment établies pour que nous cherchions à les admettre au sein de notre formalisme pourtant hérité des modèles avec relations temporelles.

Sans prétendre nous affranchir totalement de cette incompatibilité, nous avons introduit dans notre formalisme la possibilité de définir des structures dont la temporalité respecte soit un modèle avec relations temporelles, soit un modèle de type états/transitions. C'est précisément le rôle du type t. Dans le cas *linéaire*, toutes les instances des objets enfants sont explicitement représentées, manipulables et peuvent être organisées au travers de relations temporelles. Dans le cas *logique*, un nombre arbitraire, éventuellement indéfini, d'occurences d'un objet enfant est associé à une représentation de cet objet. Il est possible de décrire les enchaînements entre les instances des objets enfants grâce à des relations logiques.

Bien que nous développerons cette caractéristique par la suite, notons dès à présent que les objets simples, de part leur aspect atomique sont nécessairement de type *linéaire*.

5.4.1 Structures linéaires

Définition 5.9 *Une* **structure linéaire** *est un objet complexe dont la représenation temporelle interne est basée sur une ligne de temps unidirectionnelle et des relations temporelles.*

Chaque structure linéaire dispose donc d'une ligne de temps propre dont l'origine coïncide avec le début de la structure. Les objets enfants d'une structure linéaire viennent se positionner sur cette ligne de temps unidirectionnelle ; un *time-stamp* exprimé dans le quantum de la structure est associé au début de chaque enfant de celle-ci. Seul le début d'un enfant est daté, sa fin n'en dispose pas et est datée au travers des points de contrôle de l'enfant.

Pour illustrer nos propos sur les structures linéaires, nous présentons un exemple d'organisation hiérarchique sur la figure 5.3(a). Cet exemple implique deux structures linéaires (S_1, S_2) et trois objets simples (E_1, E_2 et E_3).

On obtient pour cet exemple les ensembles d'enfants $\mathscr{E}$ suivants :

$$\begin{aligned} \mathscr{E}(S_1) &= \{E_1@t_{e_1}, E_2@t_{e_2}, S_2@t_{s_2}\} \\ \mathscr{E}(S_2) &= \{E_3@t_{e_3}\} \end{aligned}$$

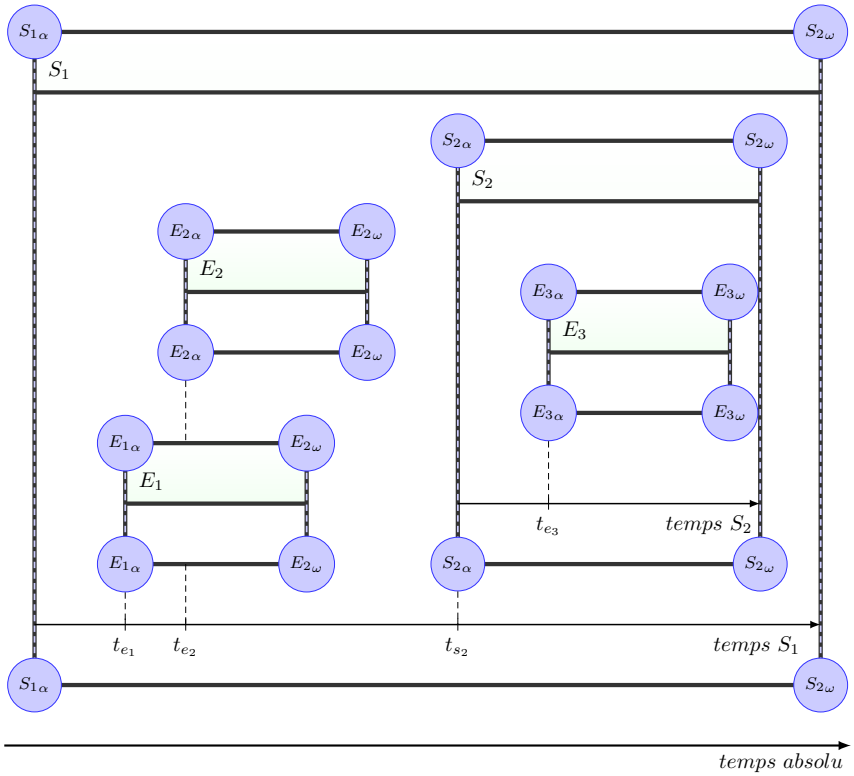
Les ensembles $\mathscr{E}$ des objets simples sont naturellement vides.

Opérateurs temporels

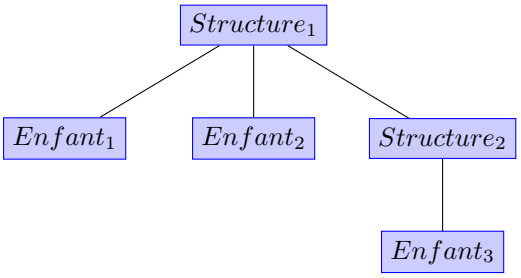
Dans le contexte des structures linéaires, il est possible de définir quelques opérateurs courants des représentations temporelles hiérarchiques comme par exemple l'opérateur *start_date* qui calcule la date du début d'un objet O relativement au début d'une structure S, exprimée dans le quantum temporel de S. Cet opérateur ne retourne un résultat que si O se trouve dans le sous-arbre ayant S comme racine. Il s'agit donc de faire la somme des *time_stamp* le long du chemin qui mène de S à O. L'algorithme 1 est une version récursive de ce calcul utilisant l'organisation arborescente des partitions.

On a alors simplement :

$$start\_date(O, S) = date\_rec(O, S, 0)$$



(a) Un exemple de structure temporelle



(b) La structure arborescente correspondante

FIG. 5.3 – Un exemple de structure temporelle

Algorithme 1 : date_rec(O,S,time_offset)

si $O==S$ **alors**
  0
sinon
  si $O==racine$ **alors**
    nil
  sinon
    si $S==parent(O)$ **alors**
      time_stamp(O)+time_offset
    sinon
      date_rec(parent(O),S, $\frac{1}{\text{r(parent(O))}}$.(time_stamp(O)+time_offset))
    fin
  fin
fin

La fonction *start_date* nous permet de définir une fonction *date* renvoyant la date d'un point de contrôle d'un objet temporel relativement à une structure :

Définition 5.10 *Soit O un objet temporel et pc_i un point de contrôle de O, on a :*

$$date(pc_i, S) = date\_rec(O, S, d_i)$$

avec d_i la date du point de contrôle relativement au début de l'objet.

Nous considérons à partir de maintenant que la fonction *date* est définie sur l'ensemble des événements de la partition, en utilisant la relation qui lie un événement et le point de contrôle qu'il représente.

Il est naturellement possible de calculer les dates des événements d'une partition dans le temps absolu. Ces dates se calculent à partir des dates dans la structure racine.

$$date(O, absolu) = r(racine).date(O, racine)$$

5.5 Les relations temporelles

Dans le cadre d'une structure linéaire, il est possible de préciser explicitement l'organisation temporelle des objets enfants de la structure à l'aide de relations temporelles binaires définies entre les objets. Ces relations temporelles remplissent deux fonctions. La première est une fonction *déclarative* qui constitue une aide à la composition lors de l'écriture de la partition. Elles permettent alors au compositeur de préciser : *"Je veux qu'il existe telle relation entre ces objets"*. Les relations temporelles sont ici utilisées comme représentation d'une volonté musicale du compositeur. L'autre fonction remplie par les relations temporelles pourait être qualifiée de *descriptive*. Dans ce cas, elles représentent les propriétées temporelles intrinsèques des objets manipulés, c'est à dire les contraintes temporelles intrinsèques des processus associés aux objets (ensemble Γ_t). Dans tous les cas, le discours musical du compositeur devra respecter les contraintes intrinsèques des processus qu'il implique.

Définition 5.11 *Soit* S *une structure linéaire, une relation temporelle* rt *de l'ensemble $\mathscr{R}(S)$ est un quintuplet :*

$$rt =< t, \sigma_1, \sigma_2, \Delta_{min}, \Delta_{max} >$$

où :

- t *est un type*
- σ_1 *et* σ_2 *sont des événements de* S

- Δ_{min} *et* Δ_{max} *sont des durées. On a :*

$$\Delta_{min}, \Delta_{max} \in \mathbb{R}_+ \cup \{\infty\}$$

et

$$\Delta_{min} \leq \Delta_{max}$$

Il y a deux types de relations temporelles :
– **pre** : relation de précédence
– **post** : relation de postériorité
En rappelant que la fonction *date* est définie sur l'ensemble des événements d'une structure S, une relation $(pre, \sigma_1, \sigma_2, \Delta_{min}, \Delta_{max})$ de S impose la contrainte suivante :

$$\Delta_{min} \leq date(\sigma_2, S) - date(\sigma_1, S) \leq \Delta_{max}$$

une relation $(post, \sigma_1, \sigma_2, \Delta_{min}, \Delta_{max})$ impose la contrainte suivante :

$$\Delta_{min} \leq date(\sigma_1, S) - date(\sigma_2, S) \leq \Delta_{max}$$

On peut noter qu'une relation $(pre, \sigma_1, \sigma_2, \Delta_{min}, \Delta_{max})$ est équivalente à une relation $(post, \sigma_2, \sigma_1, \Delta_{min}, \Delta_{max})$.
Pour étendre le vocabulaire des partitions, nous définissons la notion d'*intervalle* associé à une relation temporelle.

Définition 5.12 *L'intervalle* I *associé à une relation* $rt =< t, \sigma_1, \sigma_2, \Delta_{min}, \Delta_{max} >$ *d'une structure* S *est la durée séparant les dates de* σ_1 *et* σ_2.

$$I = date(\sigma_1, S) - date(\sigma_2, S)$$

On peut distinguer différents types d'intervalles en fonction des valeurs de Δ_{min} *et* Δ_{max} *:*
– *si* $\Delta_{min} = \Delta_{max} = 0$ *il s'agit d'une relation de* synchronisation
– *si* $\Delta_{min} = \Delta_{max} = \Delta \neq 0$ *on parle d'intervalle* rigide
– *si* $\Delta_{min} = 0$ *et* $\Delta_{max} = \infty$ *on parle d'intervalle* souple
– *sinon l'intervalle est dit* semi-souple *(ou* semi-rigide *selon l'optimisme du locuteur)*

5.5.1 Structures logiques

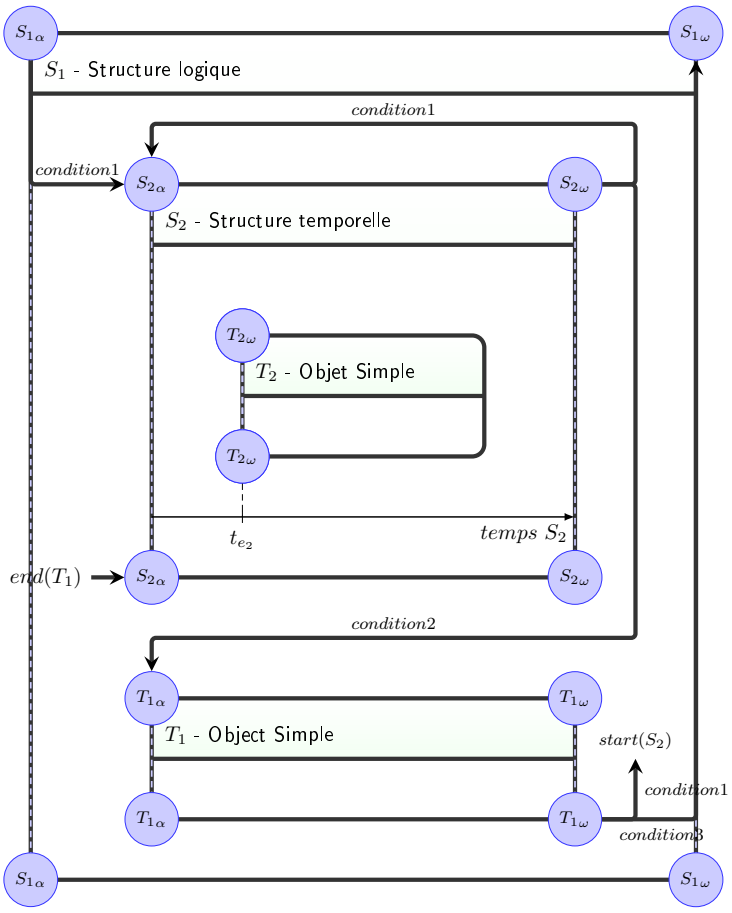
Définition 5.13 *Nous désignerons par* **structure logique** *un objet complexe dont la temporalité est basée sur un modèle type états/transitions.*

L'écriture du temps d'une structure logique s'effectue en définissant explicitement les états par lesquels doit passer la structure et les modalités de transition entre ces états. Il n'y a pas de restrictions sur la spécification des transitions ce qui permet de définir des "retours en arrière" (un enchainement **ABA** par exemple) ou encore des boucles. Pour préciser les possibilités d'expression des structures logiques, nous nous appuyons sur les automates finis auxquels elles sont assimilables dans une première approche. Nous présentons sur la figure 5.4, un exemple de structure logique ainsi que l'automate fini équivalent.

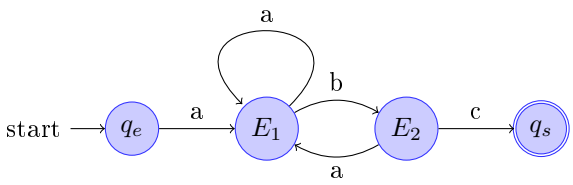
A la différence des structures temporelles et comme pour les automates, la représentation d'un objet enfant d'une structure logique peut être associée à une multitude, voire à un nombre indéfini, d'occurences de l'objet. Dans ces conditions, une structure logique ne dispose pas de ligne de temps unidirectionnelle et ses objets enfants ne sont pas assortis d'un *time_stamp*. On a ainsi pour l'exemple de la figure 5.4 les ensembles d'enfants $\mathscr{E}$ suivants :

$$\begin{aligned} \mathscr{E}(S_1) &= \{T_1, S_2\} \\ \mathscr{E}(S_2) &= \{T_2@t_{e_2}\} \end{aligned}$$

L'ensemble $\mathscr{E}(\mathscr{S})$ contient des objets non datés, ce qui ne l'empêchent pas de contenir des structures temporelles comme S_2. La représentation hiérarchique arborescente présentée précédement est donc généralisable aux partitions mélant structures temporelles et logiques. Pour rendre cette généralisation possible, nous adoptons la même convention que pour la représentation graphique sous forme de boites :



(a) Un exemple de structure logique



(b) L'automate fini correspondant

FIG. 5.4 – Un exemple de structure logique

un nœud ou une feuille représentant un objet enfant d'une structure logique représente a priori un nombre indéfini d'occurences de cet objet.

Cependant l'opérateur *date* se trouve modifié et l'absence de *time_stamp* associés aux objets enfants d'une structure logique empêchent de le définir comme précédement. Plus précisemment, $date(\sigma, S)$ n'est plus défini si il existe un nœud représentant une structure logique sur le chemin de S à l'objet portant σ. L'algorithme 2 présente la modification de l'algorithme précédent pour généraliser le calcul de *date_rec*.

Algorithme 2 : date_rec(O,S,time_offset) version générale

si *S==O* **alors**
  0
sinon
  si *O==racine ou type(parent(O))==logique* **alors**
    nil
  sinon
    si *S==parent(O)* **alors**
      time_stamp(O)+time_offset
    sinon
      date_rec(parent(O),S, $\frac{1}{r(parent(O))}$.(time_stamp(O)+time_offset))
    fin
  fin
fin

Le calcul de *date* est inchangé.

Une structure logique représente donc une évolution de la partition à travers différents états selon des enchainements prédéfinis. Dans une première approche, on peut représenter une structure logique sous la forme d'un automate fini possédant un état d'entrée, un état terminal correspondant à la fin de la structure et un état par objet enfant de la structure logique. Dans l'exemple de la figure 5.4, la structure logique S_1 est donc équivalente à l'automate présenté sur la figure reconnaissant le langage défini par l'expression rationnelle e :

$$e = a(a, ba)^*bc$$

La spécification des transitions entre états se fait au moyen de relations logiques que nous présentons dans la section suivante.

En dehors de q_e et q_s, les états de l'automate étant les objets enfants de la structure, lors du déroulement de celle-ci, l'accès à l'état représentant un objet provoque le déclenchement d'une occurence de ce dernier. Ce mode de fonctionnement pose la question de la durée d d'une structure logique. Comme nous l'avons déjà dit, les objets simples sont de type temporel disposant d'une durée clairement établie. Une structure logique contient donc des objets de durée connue dont on ne peut déterminer le nombre d'occurences réalisées pendant le déroulement de la structure. La durée d'une structure est par conséquent a priori indéterminable. Cependant les relations logiques qui permettent l'organisation des enfants d'une structure logique disposent de choix par défaut. Cela signifie que pour chaque état de l'automate, en l'absence de choix explicite, une transition privilégiée est empruntée. De plus, on impose qu'en partant de l'état initial de l'automate, la suite des choix par défaut forme un mot reconnaissable par l'automate pour assurer la fin du déroulement de la structure. La durée d d'une structure logique correspond à la durée du déroulement par défaut. La durée écrite sur la partition n'est donc pas celle de la structure lors de l'exécution de la partition.

Ces notions sont développées dans les parties concernant les relations logiques et l'exécution des pièces.

L'impossibilité de définir des dates pour un objet ne signifie par pour autant que le temps soit totalement absent des structures logiques. Preuve en est, les structures logiques peuvent impliquer des objets temporels pour lesquels des lignes de temps sont clairement définies. Ainsi, du temps s'écoule pendant l'exécution des objets enfants d'une structure logique et même éventuellement entre les exécutions des objets enfants. L'impossibilité de définir la fonction *start_date* pour les objets logiques n'est pas la conséquence d'une absence de ligne de temps mais plutôt du non-déterminisme de cette ligne de temps, cette

non-linéarité étant elle-même due à la possibilité d'exécuter un nombre indéfini d'occurences des objets enfants.

Pour pouvoir raisonner partiellement sur la temporalié des objets logiques, il faut être capable d'identifier les occurences des objets enfants d'une structure logique. Nous définissons pour ce faire une fonction *date_occ*.

Définition 5.14 *Soit Sl une structure logique, m un mot reconnaissable par l'automate correspondant à Sl et e un événement rattaché à un objet enfant du sous-arbre de Sl, date_occ*(O, Sl, i) *représente la date de la* i^{eme} *occurence de e relativement au début de Sl.*

On peut naturellement généraliser cette fonction à l'ensemble des structures en considérant que les objets d'une structure linéaire n'ont qu'une seule occurence.

Remarque 5.4 *On pourrait penser que la définition de date_occ offre la possibilité de raisonner temporellement indifféremment sur des structures temporelles ou logiques. Mais la définition que nous donnons de date_occ n'est pas suffisament formelle pour constituer les bases d'un raisonnement solide. Une piste éventuelle serait de s'intéresser à des mots sur les automates associés aux structures logiques pour caractériser la succession d'exécutions d'enfants. Une occurence d'un objet enfant serait alors identifiée non plus par un indice abstrait mais par un mot d'automate ayant conduit du début de la structure logique jusqu'à elle. Cependant, cette approche ouvre un chantier immense qui reste en dehors de l'étude présente ; nous en présenterons quelques prémices lorsque nous aborderons les perspectives de notre travail. En revanche, la définition des dates d'occurence nous sera utile pour formaliser certaines propriétés temporelles des partitions.*

5.6 Les relations logiques

Les relations logiques permettent de spécifier des enchaînements entre les états d'une structure logique. Précisons quelque peu la modèle des objets logiques avec des automates, nous affirmions que ces objets sont assimilables à des automates finis. Cette comparaison est réductrice dans la mesure où un état de l'automate correspond à l'exécution d'un objet enfant de la structure. Par conséquent, du temps s'écoule dans chaque état de l'automate et il n'est pas possible de quitter un état avant d'avoir atteint la fin de l'objet s'exécutant ; dans le cadre des partitions statiques, cette fin n'est atteinte qu'après déroulement de l'objet en question. En outre, les transitions entre les états sont définies en fonction de caractéristiques internes de ces partitions puisqu'aucune intervention extérieure n'est possible pendant l'exécution. Enfin, pour éviter des situations de blocage, des intervalles de temps sont associés aux transitions pour permettre la poursuite de l'exécution de la partition en forçant une transition par défaut si aucune des transitions n'est possible. Ces objections exprimées, nous utiliserons encore dans certaines occasions le modèle des automates d'états finis par soucis de clareté et concision.

Rappelons ici que les événements d'une structure logique sont son début, sa fin et les débuts et fins de tous ses objets enfants. Une relation logique va permetre de définir quels états (quelles exécutions d'objets) sont accessibles depuis un état donné ainsi que sous quelles conditions chacun des états suivants est accessible. Comme les changements d'état correspondent à des exécutions des objets enfants de la structure, les relations logiques sont définies entre la fin d'un objet enfant et les débuts de plusieurs autres. Les exceptions à cette règle sont d'une part les relations logiques entre le début de la structure et les débuts de certains des objets enfants pour préciser quels objets sont exécutables dès que la structure est activée. D'autre part, de manière symétrique, il est possible d'impliquer la fin de la structure dans des relations logiques ayant pour origine la fin d'un de ses objets enfants pour indiquer la désactivation de la structure après la fin d'un de ses enfants. Notons qu'il est possible de définir des boucles en spécifiant une relation logique qui implique la fin d'un objet et son début.

Comme notre modèle temporel implique que le début d'un objet précède toujours sa fin, il n'est pas possible de spécifier une relation logique aboutissant directement à la fin d'un objet enfant et comme il faut attendre la fin de l'exécution d'un objet pour changer d'état, on ne peut pas préciser de relation ayant pour origine le début d'un objet enfant.

Définition 5.15 *Soit Sl un objet logique, une relation logique rl de SL est un quadruplet :*

$$rl =< \sigma_o, \Omega, \Delta_{min}, \Delta_{max} >$$

où :

- *σ_o est un événement*
- *Ω est une liste de couples "événement, condition"*
- *Δ_{min} et Δ_{max} sont des durées*

Description

σ_o **:** l'événement *origine* de la relation. Il s'agit soit du début la structure OL, soit de la fin d'un des objets enfants de SL ($\sigma_o = end(O_o)$ ou $\sigma_o = start(SL)$).

Ω **:** une liste de couples $< \sigma_i, Cond_i >$. Les σ_i sont des événements de Sl et les $Cond_i$ des expressions logiques impliquant les variables de l'ensemble $\mathscr{V}$ de Sl. Les événements σ_i sont les débuts d'objets temporels ($\sigma_i = start(OT_i)$) ou la fin de la structure ($\sigma_i = end(SL)$). Les couples $< \sigma_i, Cond_i >$ ont la signifaction suivante :

$$Soit\ rl =< \sigma_o, \Omega >\ avec\ \sigma_o = end(OT_o)\ (resp.\ start(SL))$$

$$Si\ (\sigma_i, Cond_i) \in \Omega\ avec\ \sigma_i = start(OT_i)\ (resp.\ end(SL)),$$

alors le changement d'état du système qui consiste à déclencher OT_i (resp. terminer Sl) aprés la fin de OT_o (resp. le début de Sl) est possible si la condition $Cond_i$ est vérifiée.

$\Delta_{min,max}$ **:** les durées Δ_{min} et Δ_{max} permettent de préciser une plage de validité temporelle pour la relation logique. Pour exprimer cette notion de manière assez semblable à l'expression des relations temporelles on notera k l'indice d'occurence de l'événement σ_o et k_i l'indice d'occurence de l'événement σ_i si celui-ci est choisi après la k^{ieme} occurence de σ_o, alors la relation rl impose la contrainte :

$$\begin{aligned}&\forall k \in [\![1;\infty]\!], \forall i \in [\![1;n]\!],\\ &\quad \Delta_{min} \leq date\_occ(\sigma_i, Sl, k_i) - date\_occ(\sigma_o, Sl, k) \leq \Delta_{max}\end{aligned}$$

Ainsi, aucun changement d'état n'est possible avant $date\_occ(\sigma_o, Sl, k) + \Delta_{min}$, et si la date $date\_occ(\sigma_o, Sl, k) + \Delta_{max}$ est atteinte sans qu'aucun changement d'état ne fut possible, un changement d'état par défaut est effectué. Par définition, le changement par défaut consiste à déclencher l'événement σ_1, premier événement de la liste Ω et ce même si la condition $Cond_1$ n'est pas vérifiée. Par conséquent, il est souhaitable que la non validité de la condition $Cond_1$ n'empêche pas le bon déroulement de la partition après le choix du déclenchement de σ_1.

Comme lorsque nous définissions $date\_occ$, on peut noter ici que le calcul de k_i en fonction de k n'est pas nécessairement chose aisée.

Des esprits aventureux peuvent reconnaître dans une relation logique l'embryon d'un raisonnement temporel au sein des structures logiques, une relation logique impliquant des relations de précédence entre les occurences de l'événement origine et celles des événements cible, en notant $occ(\sigma, k)$ la k^{ieme} occurence de l'événement σ :

$$\begin{aligned}&< \sigma_o, \{\sigma_i, Cond_i\}, \Delta_{min}, \Delta_{max} > \Longrightarrow\\ &\quad \forall (k,i) \in [\![1;\infty]\!] \times [\![1;n]\!],\\ &\qquad < pre, occ(\sigma_o, k), occ(\sigma_i, k_i), \Delta_{min}, \Delta_{max} >\end{aligned}$$

Ces remarques permettraient de raisonner temporellement sur les structures logiques, mais la caractérisation des k_i nécessite d'être approffondie avant d'atteindre une homogénéisation sans douleur des deux types de structures.

Une propriété importante des structures logiques est que la suite des choix par défaut d'une structure doit mener à la fin de cette structure. En effet, l'objectif des choix par défaut est de poursuivre le déroulement de la partition lors de situations critiques, par conséquent il est nécessaire que ceux-ci conduisent à sortir de la structure logique. On peut exprimer plus formellement cette propriété en reprenant la métaphore des structures logiques avec les automates d'états finis.

Sans alourdir le propos par la formalisation de la construction de l'automate associé à une structure logique, on peut intuitivement la décrire de la manière suivante :

- créer un état par objet enfant, étiqueté avec l'objet en question
- créer un état initial étiqueté avec $start(Sl)$ et un état final étiqueté avec $end(Sl)$. A la différence des états associés aux objets enfant, ces deux états spécifiques sont étiquetés avec des événements. Ces deux états servent d'entrée et de sortie pour l'automate.
- pour chaque relation logique rl de Sl, créer une transition par événement cible, depuis l'état étiqueté avec l'objet contenant l'événement origine (ou avec $start(Sl)$), vers l'état étiqueté avec l'objet contenant l'événement cible considéré (ou $end(Sl)$). Ces transitions étant elles-mêmes étiquetées avec les relations logiques $Cond_i$.

Le mot constitué de la suite des expressions logiques associées aux choix par défaut en partant de $start(Sl)$ doit être reconnaissable par cet automate.

Il est important de noter que notre modèle n'impose pas de choix prédéfini dans le cas où plusieurs événements cible sont déclenchables à partir d'un événement. Le choix entre ces diverses possibilités est a priori indéfini. La liste Ω n'est pas triée et seul son premier élément est caractérisé.

En ce qui concerne la nature précise des expressions logiques, nous y reviendrons lorsque nous aborderons les variables de l'ensemble $\mathscr{V}$ des structures.

Précisons enfin quelques raffinements possibles concernant les relations logiques : l'ensemble des conditions $\{Cond_i\}$ peut être remplacé par un choix aléatoire parmi les σ_i. L'automate associé à la structure devient probabiliste et il est alors possible de spécifier des algorithmes de même nature avec les structures logiques. A ce propos, il devient assez clair que les structures logiques vont permettre de décrire des processus dont le calcul associé implique un algorithme mettant en œuvre des structures conditionnelles ou non déterministe. Nous présenterons d'ailleurs quelques exemples un peu plus loin.

5.6.1 Les structures comme représentation de processus complexes

L'idée selon laquelle les structures permettent de construire des algorithmes non représentables par des processus d'objets simples est sous-jacente depuis quelques pages, et il est temps de la mettre au jour. Rappelons qu'une structure ne possède aucun processus génératif ou contextuel associé.

Les caractéristiques de ce "processus de structure" ne sont pas les mêmes que celles d'un processus génératif associé à un objet simple. Pour une structure linéaire, si le nombre de ses évenements est fixé (toutes les occurences des objets enfants de la structure sont représentées), l'ordre dans lequel ils vont se produire n'est a priori pas fixé. Pour une structure logique, c'est la réalisation d'un nombre fixe d'événements qui n'est plus assurée (un nombre indéterminé d'occurences des objets enfants de la structure peuvent se réaliser). On commence à entrevoir la possibilité de représenter des algorithmes non déterministes en utilisant des structures.

Dans le cadre de ce modèle, les événements d'une structure sont considérés comme les étapes du calcul du processus complexe qu'elle représente car ceux-ci sont des moments caractérisés du déroulement de la structure. Il est par conséquent assez naturel de chercher à définir des points de contrôle pour une structure, liés à certains de ses événements.

Nous devons cependant émettre une objection en ce qui concerne les structures logiques. Leurs événements représentant un nombre indéterminé d'occurences d'eux-même, il est impossible d'associer un point de contrôle à un événement. Seuls le début et la fin de la structure, qui représentent une unique occurence sont associables à des points de contrôle.

Nous étendons alors la fonction *control* aux structures de la manière suivante :

- Cas logique :

$$control : \mathscr{P}(S) \rightarrow \{start(S), end(S)\}$$

– Cas linéaire :

$$control : \mathscr{P}(S) \rightarrow \Sigma(S)$$

Plaçons nous dans le cas d'une structure linéaire.

A la différence des étapes de calcul d'un processus, les événements d'une telle structure sont systématiquement datées. Les points de contrôle de la structure doivent naturellement retranscrire cette situation.

Propriété 5.2 *Soit S une structure linéaire, $\mathscr{P}$ l'ensemble de ses points de contrôle on a :*

$$\forall pc \in \mathscr{P}, date(pc, parent(S)) = date(control(pc, parent(S)))$$

Quelques remarques tirées de la propriété précédente. Tout d'abord, la structure racine ne possédant pas de structure parent, elle ne possède pas de points de contrôle autres que son début et sa fin. Dans le cas des structures logiques, nous pourrions éventuellement permettre la définition de point de contrôles intermédiaires, en considérant que ceux-ci représentent plusieurs occurences d'eux même ou qu'ils sont associés à toutes les occurences de l'événement auquel ils sont liés. Cette perspective rejoint la définition de *date_occ* et les tentatives de raisonnements temporels généralisés dans le futur de nos recherches en cours.

Cette propriété montre également que l'intérêt des points de contrôle des structures est aussi de représenter ses événements dans les niveaux de hiérarchie supérieurs et de permettre une écriture du temps impliquant des événements de différents niveaux hiérarchiques. Finalement, ceci correspond bien à l'analogie faite entre les structures avec leurs "processus complexes" et les objets simples. Une structure permet la mise en temps d'événements de bas niveau dans une temporalité plus large. Cette dernière assertion fait écho à la remarque formulée précédemment au sujet de la finesse de représentation des propriétés temporelles d'un processus, une structure constitue un niveau d'abstraction supplémentaire dans la construction de la partition depuis le niveau sous-symbolique. Une structure ne conserve alors la représentation de certaines propriétés temporelles, nécessaires à l'écriture au niveau hiérarchique dans lequel elle se trouve impliquée.

Les éventuelles relations temporelles définies entre les événements d'une structure peuvent alors être vues comme des contraintes intrinsèques de celle-ci. Nous développons ces notions au chapitre suivant.

5.6.2 Reflexion autour de l'orientation du temps dans les partitions

Dans les paragraphes précédents, nous avons évoqué quelques caractéristiques de notre modèle qui méritent que l'on s'arrête un peu plus longuement. En particulier, le fait que nous considérions les objets simples comme "atomiques" donc nécessairement de type *linéaire*, et que leurs points de contrôle sont dans un ordre déterminé limitent clairement les possibilités du modèle. Ceci implique qu'au niveau le plus bas de la hiérarchie, la flèche du temps est toujours dirigée dans le même sens (l'ordre chronologique écrit des points de contrôle). Quel que soient les modifications de l'ordre de déroulement des objets simples au moment de l'exécution que permettent l'organisation dans des structures, les "briques élémentaires" que consituent les processus seront toujours exécutées de la même manière. Il est par exemple impossible d'exécuter un processus en ordre inverse pour, par exemple, jouer un son à l'envers directement à partir du processus effectuant la lecture de ce son à l'endroit. A plus fortes raisons, des modifications encore plus complexes de la ligne du temps telles que celles offertes par le logiciel Iannix sont hors de notre portée. Bien sûr, il est possible d'approcher ce type de comportement en découpant les processus en sous-processus de tailles minimales et en les organisant dans des structures pour permettre leur exécution dans des ordres variables. Cependant, cette possibilité reste cantonée à quelques cas bien particuliers. A titre d'exemple, appliquer cette méthode pour permettre la lecture d'une courbe à l'envers nécessiterait de séparer la lecture de chaque échantillon en un processus particulier, de les organiser dans une structure temporelle et piloter explicitement le déclenchement de chacun d'entre eux dans l'ordre voulu. Avec les fréquences d'échantillonage communément utilisées dans les processus musicaux (supérieure à 40000Hz), cette solution n'est clairement pas réaliste.

Cette limite est une conséquence de l'approche que nous avons suivie pour aboutir à ce modèle, à savoir l'interprétation d'un matériel musical "pré-écrit" qui nécessite un certain déroulement depuis

le début de la pièce jusqu'à son terme. Le choix de nous appuyer sur les modèles hiérarchiques avec relations temporelles est également responsable de cette limitation, tant les raisonnements sur des relations temporelles s'appuient sur l'existence d'une ligne de temps unidirectionnelle.

On pourrait éventuellement pallier cette limite en modifiant les processus utilisés. En reprenant l'exemple de la lecture de table, il serait éventuellement possible de paramêtrer ce processus pour lui préciser lors de son éxecution un indice où commencer la lecture de la table et un sens de lecture. Cependant, il s'agit là de modifications au niveau du calcul du processus, l'ordre de déroulement des points de contrôle de l'objet représentant l'exécution de ce processus serait toujours le même.

5.7 Les branchements

Les branchements de l'ensemble $\mathscr{B}$ d'une structure permettent de diriger les flux de sortie d'objets enfants vers les entrées d'autres objets ou encore vers les sorties de la structure elle-même, pour permettre la communication des données vers les niveaux hiérarchiques supérieurs.

Définition 5.16 *Soit St une structure, un branchement de l'ensemble $\mathscr{B}(S)$ est un couple :*

$$b =< io_1, io_2 >$$

dans lequel, io_1 et io_2 vus comme des variables sont des entrées sorties de la structure St elle même ou de ses objets enfants :

- $io_1 \in In(St) \cup \{ \bigcup_{O \in \mathscr{E}(St)} Out(O) \}$, *la source de b*
- $io_2 \in Out(St) \cup \{ \bigcup_{O \in \mathscr{E}(St)} In(O) \}$, *la destination de b*

Nous n'interdisons pas a priori le branchement d'une sortie d'un objet avec une entrée de ce même objet, ni même la définition de plusieurs branchements vers une même destination.

Il faut voir les entrées et sorties des objets temporels pour ce qu'ils sont précisément, c'est à dire les représentations des arguments et résultats des calculs algorithmiques des processus associés aux objets. On peut ainsi les considérer comme des symboles représentant des variables auxquelles une valeur est assignée à un moment donné. Naturellement, les valeurs changent au cours de l'exécution de la partition.

Un branchement peut donc être considéré comme un appel fonctionnel vu comme la connection entre l'argument d'un calcul et le résultat d'un autre calcul. La particularité étant bien sûr que cette connection est temporisée.

Définition 5.17 val *est la fonction qui renvoie la valeur assignée à une variable, cette fonction renvoie nil si la variable n'est pas assignée.*

Propriété 5.3 *Soit $b =< io_1, io_2 >$ un branchement, on a :*

$$val(io_1) = x \;\Rightarrow\; val(io_2) = x$$

Passons sous silence pour l'instant les épineux problèmes de pénurie de valeurs d'entrées ou de conflits entre différentes valeurs, posés par cette définition des branchements. Ces considérations restent largement en dehors de notre étude, cependant nous soulèverons un coin du voile par la suite en abordant les relations temporelles implicites liées aux branchements.

Le point important à retenir ici est l'approche des entrées et sorties des objets considérés comme des variables auxquelles on peut assigner des valeurs, les branchements étant des modes d'assignation des variables. La formulation de cette remarque juste avant d'entamer la présentation des variables est très peu redevable du hasard.

5.8 Les variables

Nous venons de le voir, l'intérêt d'une variable est de se voir assignée une valeur susceptible d'être modifée et d'être mise en relation avec les valeurs d'autres variables. L'ensemble $\mathscr{V}$ d'une structure définit un environnement pour cette stucture. Les valeurs des variables qu'il contient permettent de capturer l'état dans lequel se trouve la structure. Naturellement la question majeure est : "Quelles variables sont pertinentes pour décrire l'état courant d'une structure ?" La réponse de bon sens est certainement : "Tout dépend de la structure considérée." Sans nier totalement le relativisme dont est empreint cette assertion, il est possible de formuler un certain nombre de points communs entre les structures et d'en déduire des variables qui fassent sens pour la majorité d'entre elles.

Nous distinguons deux types de variables, d'une part les variables *temporelles*, qui sans surprise représentent des caractéristiques liées au temps et à son écoulement, et d'autre part l'ensemble défini en creux des variables *non-temporelles*.

Notons que les variables d'une structure n'ont réellement de sens qu'accompagnées des contraintes que l'on peut définir entre elles et que nous présentons à la suite. En effet, tout l'intérêt de pouvoir spécifier des variables pour une structure est de pouvoir ensuite décrire les relations qu'elles entretiennent les unes avec les autres, ainsi qu'avec les calculs des processus.

5.8.1 Les variables temporelles

Notre formalisme étant construit avec comme briques élémentaires des "objets temporels", il est certain que les caractéristiques temporelles se retrouvent dans toutes les structures. Les variables temporelles ont un statut particulier dans notre modèle.

Quantum temporel

Le quantum de temps qui sert de base au calcul des dates et durées au sein d'une structure et que l'on retrouve dans toutes les structures est une variable temporelle de chaque structure. Cependant, pour respecter l'organisation hiérarchique des partitions, nous utilisons plutôt le rapport $r(S)$.

Propriété 5.4 *Soit St une structure, on a :*

$$r(St) \in \mathscr{V}(St)$$

Nous verrons qu'utiliser une variable pour l'unité temporelle d'une structure permet de définir des opérations de modification globale des dates des événements de la structure.

5.8.2 Variables d'intervalle

La possibilité de définir clairement les dates des occurences des événements au sein des structures linéaires, permet d'associer des variables aux intervalles de temps séparant ces événements.

Propriété 5.5 *Soit une structure linéraire Sl :*

$$\forall(\sigma_1,\sigma_2) \in \Sigma(Sl)^2, \exists I \in \mathscr{V}(Sl) \ t.q.\ val(I) = date(\sigma_1, Sl) - date(\sigma_2, Sl)$$

On peut remarquer que les dates des événements possèdent elles-mêmes des variables associées dans la mesure où :

$$\forall\sigma \in \Sigma(Sl), date(\sigma, Sl) = date(\sigma, Sl) - date(start(Sl), Sl)$$

La définition des variables d'intervalle laisse entrevoir les liens qui peuvent se tisser entre des variables puisque la définition de *date* est directement basée sur la valeur du quantum q.

Signalons également le rapport entre les relations temporelles et les variables d'intervalles, les premières venant limiter les valeurs des secondes.

Les variables temporelles ont ceci de particulier que seules celles que nous venons d'exposer peuvent être présentes dans l'environnement d'une structure. La raison en est que ces variables touchent au cœur

du modèle et que les mécanismes que nous mettons en œuvre pour exécuter les partitions en respectant les relations temporelles, nécessitent pour être efficace que les comportements temporelles des éléments des partitions soient circonscrits à ceux définis dans le formalisme. Nous reviendrons sur ce point en abordant la machine d'exécution des partitions.

Typage

Afin de rendre leur manipulation aisée, les variables sont typées. Dans la mesure où notre modèle s'appuie sur du temps continu, toutes les variables temporelles sont à valeurs dans $\mathbb{R}$.

5.8.3 Les variables non temporelles

Les possibilités de définition de variables non temporelles sont encore plus variées que celles des variables temporelles et sont très dépendantes de la nature de la structure. Cependant, on peut tout de même formuler quelques propriétés générales concernant ces dernières. Les variables non-temporelles permettent de décrire l'état d'une structure relativement à son contenu.

Le suspense ayant été désamorcé il y a quelques pages déjà, actons le fait que les entrées et sorties d'une structure et de ses objets enfants sont des variables de la structure en question :

Propriété 5.6 *Soit une structure St :*

$$E(St) \cup S(St) \cup \bigcup_{O \in \mathscr{E}(St)} E(O) \cup S(O) \subset \mathscr{V}$$

Un cas particulier de variable, liées au contenu des objets non par une valeur de paramêtre mais par une valeur bouléeene, est tout simplement celle qui indique si un objet est en cours d'exécution. L'intérêt que l'on peut trouver dans ce type de variable, nous pousse à la définir systématiquement pour les enfants d'une structure. Nous la noterons $v_{ex}(O)$.

L'imagination des compositeurs les conduit facilement à définir des variables non-temporelles, et à la différence des variables temporelles, il serait illusoire de tenter d'en dresser une liste.

Citons à titre d'exemple une variable non-temporelle plus élaborée, soufflée par Charlotte Truchet : le nombre maximum de textures qui s'exécutent au même moment. Nous intégrons cette variable au formalisme pour les liens entre le comportement temporel des partitions et les variables de contenu qu'elle illustre plus que pour son réel intérêt compositionnel. En effet, pour que cette notion soit réellement pertinente du point de vue du compositeur, il serait souhaitable de distinguer le rôle des textures et considérer uniquement celles qui produisent du son diffusé ("les notes"). Cependant, nous présentons par la suite des mécanismes permettant de gérer le lien temporel-contenu, en espérant qu'ils pourraient inspirer la manipulation de variables de contenu plus générales. Pour chaque structure, nous définissons donc une variable $nb - tex(S)$ représentant cette notion, ainsi qu'une variable connexe, $nb - tex_{max}(S)$ représentant le nombre maximum d'objets simples pouvant s'exécuter au même moment.

Typage

Comme les variables temporelles, les variables non-temporelles se doivent d'être typée. La description des processus doit ainsi contenir les types des valeurs qu'ils acceptent en entrée, ainsi que celles des valeurs qu'ils retournent. Par conséquent, les variables représentant ces valeurs, peuvent être typées en fonctions des processus en question. Enfin, la définition d'une nouvele variable non-temporelle par l'utilisateur implique son typage.

Portée et hiérarchie

Variable locale, variable globale ... Ces notions classiques en programmation commencent à poindre au sein de notre modèle au travers de la définition des variables et de l'organisation hiérarchique. Formalisons quelque peu ces notions. Nous définissons la portée d'une variable comme étant l'ensemble des structures dans lesquelles cette variable est visible et donc accessible. La portée d'une variable d'une structure S est l'ensemble des structures descendantes de S.

Propriété 5.7 *Soit St une structure,*

$$\forall S \in subtree(St), \mathscr{V}(St) \subset \mathscr{V}(S)$$

Ainsi les variables de la structure racine sont désignées comme des variables *globales* de la partition puisque accessibles depuis n'importe quelle structure de la partition. A l'inverse, les autres variables sont désignées comme des variables *locales* dans la mesure où leur portée est limitée.

On pourrait s'interroger sur la naissance de conflits autour des noms de variables. Il faut noter que de nombreuses variables parmi celles que nous avons proposées sont identifiables par leur rôle : quantum temporel d'une structure, entrée/sortie d'un objet ... Ces dernières sont implicitement nomées de part leur rôle. Seules les variables explicitement définies par l'utilisateur peuvent générer des conflits de nomage.

5.9 Les contraintes de l'ensemble $\mathscr{C}$

Nous avons déjà insisté sur ce sujet, les variables des structures son intéressantes du point de vue des relations qu'elles entretiennent entre elles. C'est à dire, du point de vue des liens entre les différentes caractéristiques des structures qu'elles décrivent. Quels sont les effets de la modification d'un paramètre interne de la partition sur ses autres paramètres ? C'est la question à laquelle répondent les contraintes de l'ensemble $\mathscr{C}$.

Une situation dans laquelle la modification de la valeur de variables se répercute sur celle d'autres variables selon des règles données, suggère un système de *pertubation*. C'est en effet l'approche que nous adoptons dans la mesure où une partition statique représente un matériau musical susceptible d'être modifié pendant son exécution celui-ci devant s'adapter à ces modifications. La question de la résolution du système à proprement dit n'est cependant pas négligée, mais elle constitue en fait une fonctionnalité de l'outil d'édition des partitions, vu comme une aide à la composition.

Les éléments de l'ensemble $\mathscr{C}$ sont donc des contraintes permettant de décrire un réseau de propagation.

Elles se définissent donc comme suit par la relation qu'elles imposent entre des variables et les mécanismes de diffusion des pertubations.

Définition 5.18 *Soit une structure St, une contrainte de $\mathscr{C}(St)$ se définit comme un triplet :*

$$c =< V, r, \mathscr{M} >$$

avec :

- *V un sous-ensemble de variables de St*
- *r une relation*
- *$\mathscr{M}$ un ensemble de méthodes*

Description

V : On a $V = \{v_i\}_{i \in [!1,n!]}$ avec v_i variable de la structure St. On notera D_i le domaine de valeurs de la variable v_i.

r : C'est la relation imposée par la contrainte c, elle est donc vraie si la contrainte est vérifiée et fausse dans le cas contraire. Formellement :

$$r \ : \ D_1 \times \ldots \times D_n \rightarrow \{vrai, faux\}$$

$\mathscr{M}$: Les méthodes de cet ensemble permettent de décrire les règles de propagation associées à la contrainte c, c'est à dire des stratégies permettant au système de réagir si une modification de la valeur d'une variable invalide la relation r.

Soit $\mathscr{M} = \{m_k\}_{k \in [\![1,m]\!]}$ on a :

$$\forall k \in [\![1,m]\!],\ m_k : D_1 \times \ldots \times D_n \rightarrow D_1 \times \ldots \times D_n$$

De plus, soit $(val_1, \ldots, val_n) \in D_1 \times \ldots \times D_n$ tel que :

$$r(v_1, \ldots, v_n) = faux$$

Sous certaines conditions,

$$\exists k \in [\![1,m]\!] \ /\ r(m_k(val_1, \ldots, val_n)) = vrai$$

Cette définition est celle classiquement utilisée pour les contraintes de pertubation mais nous n'autorisons pas la définition de n'importe quelles contraintes.

Contraintes d'intervalles

Le premier cas de contraintes que nous envisageons concerne uniquement des variables d'intervalle d'une structure linéaire et s'expriment sous forme de combinaison linéaire :

Définition 5.19 *Soient S une structure linéaire, une contrainte d'intervalle c de $\mathscr{C}(S)$ est telle que :*
- *V : $\{I_i\}_{i \in [|1,n|]}$ où les I_i sont des variables représentant la valeur d'intervalles entre des événements de S.*
- *r : $\sum_{k=1}^{n} \alpha_k . I_k = C$ avec C une constante*
- *$\mathscr{M}$ contient des méthodes correspondant à des stratégies d'adaptation que nous définissons par la suite.*

Cette forme de contraintes est la plus courante. Le compositeur les définit en s'appuyant les contraintes pré-établies que nous présentons dans le chapitre traitant des partitions interactives.

Contraintes portant sur les quanta temporels

L'utilisateur peut poser des contraintes entre les quanta temporels de structures différentes. Comme pour les intervalles, il s'agit de combinaisons linéaires.

Définition 5.20 *Soit St une structure linéaire, une contrainte sur les quanta sur des structures enfants S_k de St est telle que :*
- *V : $\{r(S_1) \ldots r(S_n)\}$*
- *r : $\sum_{k=1}^{n} \alpha_k . r(S_k) = C$ avec C une constante*
- *$\mathscr{M}$: comme pour les contraintes d'intervalles, l'ensemble de méthodes contient des stratégies prédéfinies.*

Ce type de contraintes permet par exemple de simuler des comportements particuliers comme le *rubato* que nous verrons dans le chapitre concernant les partitions interactives.

Le compositeur peut également poser des contraintes entre le quantum temporel d'une structure et une variable d'entrée/sortie de cette structure. Pour utiliser au mieux la définition hiérarchique des quanta, ces contraintes s'expriment par l'intermédiaire du rapport $r(S)$.

Définition 5.21 *Soit S une structure, v une variable d'entrée/sortie de l'ensemble $\mathscr{V}(S)$, une contrainte de vitesse de $\mathscr{C}(S)$ est telle que :*
- *V : $\{r(S), v\}$*
- *r : $r(S) = k.v$, avec k une constante.*
- *$\mathscr{M}$:*
  - *m_1 : $v \leftarrow \frac{r(S)}{k}$*
  - *m_2 : $r(S) \leftarrow k.v$*

Lier le quantum temporel d'une structure à une variable d'entrée/sortie permet de modifier la valeur de ce quantum grace à l'exécution d'un objet contextuel (méthode m_2) et ainsi de simuler aisément des changements de tempo comme nous le verrons plus loin. La méthode m_1 permet à l'inverse de modifier un paramêtre de contenu en fonction du contenu, par exemple associer la hauteur d'un son de synthèse avec la vitesse d'exécution.

Contraintes sur le nombre de textures

Deux sortes de contraintes sont définissables pour une structure S. D'une part lier la valeur de $nb-tex_{max}(S)$ avec la sortie d'un objet contextuel pour venir la modifier au cours du déroulement de la partition. Et d'autre part, lier la valeur a du nombre d'objets s'exécutant ($nb-tex(S)$) avec l'entrée d'un objet pour adapter le paramêtre d'un processus à cette valeur.

Comme nous l'avons annoncé, ces variables dévoilent une part importante des interconnexions entre le comportement temporel et les variables de contenu. Pour bien expliciter ces relations il nous faut présenter deux contraintes intrinsèques, sous-jacentes à la définition des variables sur le nombre d'objets.

On trouve d'une part la contraintes entre $nb-tex(S)$ et les variables $v_{ex}(O)$ imposant la relation :

$$nb-tex(S) = \sum_{O \in \mathscr{E}(S)} v_{ex}(O) + \sum_{St \in \mathscr{E}(S)} nb-tex(St)$$

Il s'agit de la définition de la variable $nb-tex(S)$ en intégrant la hiérarchie.

Cependant la seule méthode de propagation associée à cette contrainte consiste à modifier $nb-tex(S)$ en fonction des valeurs des $v_{ex}(O)$. En effet, la définition d'une stratégie adaptant les $v_{ex}(O)$ à une valeur arbitraire de $nb-tex(S)$ aurait des conséquences difficilement conciliables avec les relations temporelles.

En outre, il existe une contrainte entre $nb-tex(S)$ et $nb-tex_{max}(S)$ qui sans surprise impose la relation :

$$nb-tex(S) \leq nb-tex_{max}(S)$$

Cette contrainte ne dispose d'aucune méthode de propagation. Modifier $nb-tex_{max}$ en fonction de $nb-tex$ n'aurait pas grand sens et une modification de $nb-tex$ ne pourrait être répercutée d'après ce qui a été dit précédemment. Le système ne pouvant s'adapter à l'invalidation de cette relation, il convient d'empêcher qu'elle se produise.

Dans cette optique, nous proposons que le compositeur puisse définir un statut pour chaque objet :
- *décallable* : l'exécution de l'objet peut être retardée jusqu'à ce qu'elle devienne possible vis à vis du nombre maximum de textures.
- *masquable* : si l'exécution de l'objet n'est pas possible compte tenu du nombre maximum de textures autorisées, celui-ci s'exécute en muet : ses sorties ne produisent aucune valeur.

Les objets contextuels ont systématiquement un statut *masquable*. En effet, la seule possibilité pour que l'exécution d'un objet contextuel invalide la contrainte sur le nombre de textures, est que celui-ci modifie le nombre de textures autorisées, le rendant incompatible avec le nombre courant de textures s'exécutant. Si cette situation se produit, la modification de la contrainte est annulée.

Dans le cas de textures décallables, si la modification de la date de déclenchement est susceptible de contredire une relation temporelle ou une contrainte d'intervalles, la texture est déclenchée en muet.

Hiérarchie et planification

Comme pour les variables, il existe un mécanisme d'inclusion des ensembles de contraintes à travers la hiérarchie :

Propriété 5.8 *Soit St une structure,*

$$\forall S \in subtree(St), \mathscr{V}(S) \in \mathscr{V}(St)$$

Dans tout réseau de pertubation, le comportement du système face à une pertubation est défini par une planification, c'est à dire le choix d'une méthode de propagation par contrainte. Pour les partitions statiques, cette planification implique le choix d'une méthode par contrainte d'intervalle nous verrons ultérieurement quelles opportunités s'offrent à lui. Pour les contraintes sur les quanta la stratégie n'est pas modifiable :
- si v est une variable d'entrée : m_1
- si v est une variable de sortie : m_2

Ces choix correspondent exactement aux rôles joués par ces contraintes et exposés plus haut.

Remarque 5.5 *On pourra noter que nous ne proposons pas de contraintes impliquant uniquement des variables de contenu. Comme nous l'avons dit, les variables de contenu peuvent être diverses ainsi les contraintes portant sur elles peuvent elles-mêmes être très variées. Pour ne pas limiter les choix des compositeurs nous préférons ne pas proposer ce type de contraintes. Il est cependant possible d'écrire ce type de calcul au travers d'objets temporels prenant en entrées les variables contraintes et fournissant en sorties des valeurs respectant un ensemble de variables. L'écriture du processus associé laisse toute liberté au compositeur d'utiliser le système de contraintes de son choix.*

Dans ce cas pourquoi ne pas laisser cette même liberté pour les variables temporelles ? La réponse rejoint la remarque concernant l'exhaustivité des variables temporelles : celles-ci étant au cœur du formalisme, les contraintes suceptibles de les modifer sont également des point cruciaux du modèle. Ainsi pour garantir les propritétés temporelles des partitions et efficacement gérer le temps lors de l'exécution de ces dernières, qui reste le propos central de cette étude, il nous faut restreindre les contraintes temporelles au cas que nous maîtrisons.C'est pour la même raison que les contraintes mêlant variables temporelles et variables de contenu sont si limitées.

Relations temporelles et branchements

Les relations temporelles peuvent être considérées comme des contraintes sur les variables d'intervalle, puisqu'elles en limitent les valeurs. Dans ces conditions, des contraintes quantitatives sur les intervalles seraient redondantes. Cependant, il est clair qu'elles devront être prises en compte lors des résolutions du système de contraintes.

De la même manière, les branchements constituent des contraintes entre des variables d'entrée/sortie. La définition que nous en avons donnée plus haut et notamment la propriété 5.3, était directement inspirée par cette approche.

Validité temporelle des contraintes

L'exécution d'une partition au cours du temps va progressivement conduire à déterminer les valeurs des variables de manière définitive. Les dates d'occurence des événements vont fixer définitivement les valeurs des intervalles et la terminaison des objets va fixer celles de leurs variables d'entrée et de sortie. Lorsqu'il ne reste plus dans une contrainte qu'une seule variable dont la valeur n'est pas fixée, il est inutile de chercher à propager une pertubation à travers elle. Par conséquent, le réseau de contraintes se modifie pendant l'exécution de la partition, les contraintes devenant peu à peu inutiles.

Synthèse de variables et propagation différée

La validité temporelles des contraintes prend un autre visage lorsqu'on s'intéresse au moment où les valeurs de variables vont être propagées à travers le réseau de contraintes. En effet, les sections précédentes, nous avons insisté sur la visibilité des variables au travers de la hiérarchie des partitions. La présence de toutes les variables des structures constituant un sous-arbre de la hiérarchie dans l'ensemble de contraintes de la structure racine sous-arbre, pourrait laisser penser que la propagation d'une nouvelle valeur se fait simultanément dans toutes les contraintes. Ce n'est pas le cas. De la même manière qu'un processus va voir sa valeur de sortie propagée au moment de son exécution, les nouvelles valeurs des variables d'une strucuture ne sont propagées qu'au moment de leur exécution.

Précisément, si la valeur d'une variable d'une structure est modifiée pendant l'exécution de celle-ci, alors la nouvelle valeur est propagée au travers des contraintes dans lesquelles la variable est impliquée. Si la structure n'est pas en cours d'exécution, alors la nouvelle valeur n'est pas propagée mais conservée, au moment où la structure est déclenchée, la valeur est propagée. On peut noter que cette situation peut impliquer que des contraintes ne soient pas respectées pendant un certain temps.

Ce mécanisme est important pour permettre de conserver des valeurs de variables qui doivent attribuée à une variable à un instant précis de la partition. On pourra ainsi lier une même variable avec plusieurs contraintes, qui viendront modifier sa valeur successivement au cours du temps.

On trouvera un exemple d'utilisation de ce mécanisme dans le chapitre 7, dans une configuration simulant un *rubato* (section 7.2.1.0).

5.9.1 Discussion

On pourrait s'interroger sur l'opportunité d'intégrer le système de contraintes aux partitions statiques et préférer les réserver uniquement aux partitions interactives. En effet, certaines d'entre elles prennent tout leur sens lorqu'il est possible de modifier directement le contenu des partitions au cours de leur exécution.

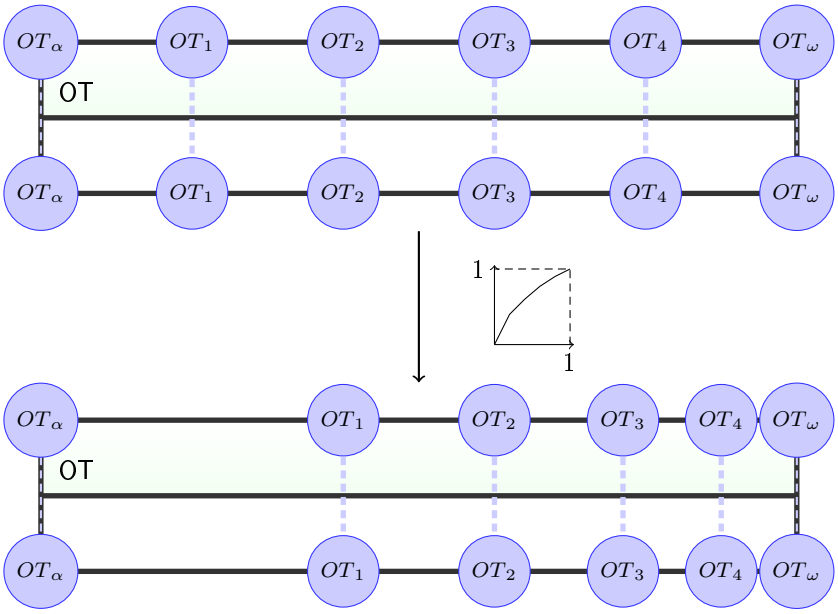
Nous pouvons opposer deux arguments à cette conception. Tout d'abord nous considérons les partitions statiques comme l'organisation d'un matériau musical incluant la description des relations entre les différents éléments de ce matériau. Ainsi, les contraintes qui sont la représentation de ces relations ont toute leur place dans les partitions statiques. Le fait de rendre une partition interactive consiste à autoriser la modification de certains éléments pendant le déroulement la partition, la réaction de la partition à ces modifications étant une donnée musicale indépendante.

De plus, introduire les contraintes dans les partitions statiques permet de modéliser facilement certains phénomènes musicaux. L'exemple le plus flagrant est certainement les variations de tempo. Pour en faire la démonstration, nous nous appuyons sur les travaux de Peter Hanappe, qui dans sa thèse [50] formalise la modification de données temporelles comme des dates d'événements au travers de ce qu'il appelle un *modèle*. Il sagit d'une fonction qui associe à des événements une nouvelle date calculée à partir de leur date courante.

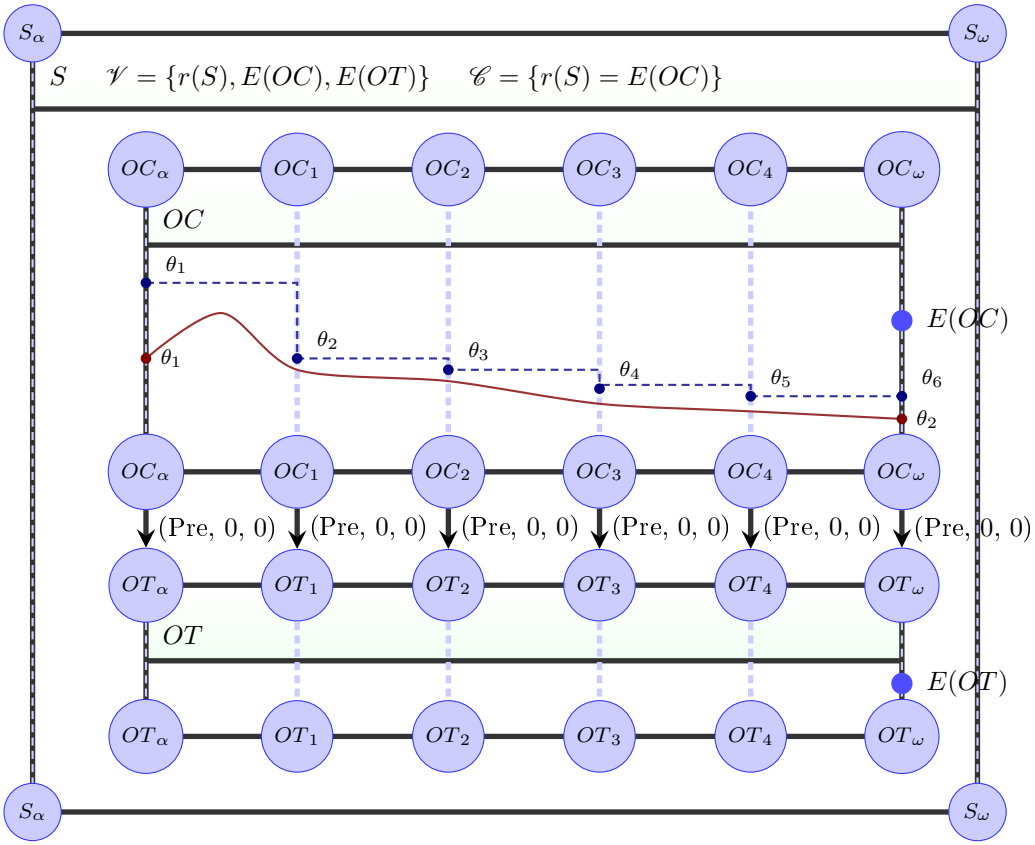
La figure 5.5(a) présente un exemple d'une telle transformation : l'objet OT voit les dates de ses points de contrôle modifier une courbe faisant correspondre à une date écrite lue en abscisse, la nouvelle date lue en ordonnée. La pente de la courbe indique si il s'agit d'une compression ou d'une extension temporelle.

La figure 5.5(b) présente une structure contenant l'objet OT ainsi qu'un objet contextuel OC modifiant le rapport $r(S)$. Deux possibilités sont envisagées pour le processus de OC, la lecture d'une courbe discrète modifiant le quantum temporel de S au déclenchement de chaque point de contrôle (en pointillé) ou la lecture d'une courbe continue.

Bien que OT soit représenté avec ses dates d'origine, l'exécution de la structure S va conduire à la modification des dates des points de contrôle par modification du quantum temporel. L'utilisation des contraintes permet ici une modélisation plus aisée et plus fine du changement de tempo.



(a) Modélisation d'un changement de tempo par modification d'objet



(b) Modélisation d'un changement de tempo par contraintes sur le quantum temporel

FIG. 5.5 – Un exemple de modélisation d'un changement de tempo

Chapitre 6

Temporalité implicite

Dans le chapitre précédent, nous avons insisté sur une approche compositionnelle des objets temporels et des relations qui viennent structurer la partition. Nous avons considéré que les relations étaient l'expression d'une ecriture musicale. Mais les relations temporelles jouent ausssi un rôle de description des proriétés intrinsèques des objets mis en œuvre dans les partitions. Dans ce cas, les relations se présentent comme des outils permettant au compositeur d'appréhender le matériau musical qu'il manipule.

Ces relations temporelles intrinsèques ou implicites ont plusieurs origines : la nature des processus impliqués, les répercutions de la construction temporelle d'une structure à travers l'organisation hiérarchique de la partition, ou encore les conséquences des branchements entre les entrées et sorties des objets. Dans chacune de ces situations, des relations temporelles entre les points de contrôle des objets permettent d'expliciter ces contraintes implicites.

Dans un contexte de composition assistée, la matérialisation de ces propriétés constitue une délimitation des possibilités du compositeur, inhérente à ses choix musicaux. D'ailleurs, le dialogue qu'entretient le compositeur avec ces limites touche d'une certaine manière au cœur du processus de composition.

6.1 Temporalité intrinsèque des processus

La cas le plus direct de relations temporelles induites, déja évoqué au chapitre précédent, concerne les propriétés des processus de la partition.

La temporalité des objets "hors-temps" que sont les processus se traduit par des contraintes temporelles entre leurs étapes de calcul. D'une part, la définition que nous avons faite des processus implique clairement un ordre total sur leurs étapes de calcul.

Ces relations induites découlant des rapport qu'entretient le modèle avec la flêche du temps sont appelées relations de *consistence*. Elles ne sont valables que pour les objets temporels **simples**.

Formulation 6.1 *Soit* OT*, un objet temporel non ponctuel possédant un ensemble de points de contrôle* $\mathscr{P} = \{pc_1, \ldots, pc_n\}$*,* $n-1$ *relations temporelles de l'ensemble* $\mathscr{R}$ *de sa structure parent découlent de l'ordre partiel entre ses étapes de calcul :*

$$\forall i \in [1, n-1], < pre, pc_i, pc_{i+1}, \Delta_{i_{min}}, \Delta_{i_{max}} >$$

Nous étendons alors le vocabulaire des *intervalles* aux objets temporels en s'appuyant sur les propriétés des intervalles associés aux relations de cohérence des objets :

– si tous ses intervalles sont *souples*, l'objet est dit *souple*.

– si tous ses intervalles sont *rigides*, l'objet est dit *rigide*.

– sinon l'objet est dit *semi-souple* (ou *semi-rigide*)

Dans le cadre d'une composition, il est bien sûr possible de modifier les dates des points de contrôle pour faire entrer le processus dans la temporalité de la partition. Ces modifications de dates s'effectuent dans les limites des relations induites.

D'autres relations temporelles sont induites des propriétés intrinsèques des processus. Il s'agit de la traduction des contraintes qualitatives entre les étapes de calcul du processus décrites dans son ensemble Γ_t. Pour pouvoir répercuter convenablement ces contraintes au niveau des relations temporelles, il convient qu'elles soient représentées dans un formalisme compatible avec celui que nous proposons pour les objets temporels. La linéarité des objets temporels impose que les contraintes temporelles entre les étapes de calculs soient vérifiées à chaque exécution. Une disjonction entre plusieurs contraintes par exemple ne pourrait être prise en compte. Ces considérations rejoignent les remarques précédentes sur la ligne de temps dans notre modèle, et l'utilisation de structures pour modéliser des processus qui ne sont pas des séquences d'étapes devant s'exécuter toujours dans le même ordre. Ceci nous permet d'insister sur l'importance de la définition d'un formalisme de description des processus en lien avec notre modèle.

6.2 Relations temporelles implicites et structures

En reprenant la définition des structures comme extension des objets simples pour la construction de processus "complexes", il parait logique qu'elles fassent également l'objet de mécanismes de répercutions de propriétés temporelles.

A la différence des objets simples, les structures sont soumises à une seule relation de consistance, celle qui assure que le début de la structure précédera toujours sa fin.

Formulation 6.2 *Soit S une structure qui n'est pas la structure racine, elle implique une unique relation temporelle dans l'ensemble $\mathscr{R}$ de sa structure parent :*

$$< pre, start(S), end(S), duration_{min}, duration_{max} >$$

La propriété de "souplesse" d'une structure est donc directement celle de son intervalle de durée.

L'absence de relations implicites entre les éventuels points de contrôle intermédiaires d'une structure linéaire, est directement liée au fait que les événements d'une structure sont partiellement ordonnés à la différence des étapes de calculs d'un processus. C'est d'ailleurs un des principaux intérêts des structures pour représenter des algorithmes non-déterministes. La figure 6.1 présente une situation simple dans laquelle l'ordre des points de contrôle d'une structure temporelle est modifié.

La structure de cet exemple peut représenter l'exécution d'un algorithme non déterministe comportant trois étapes dont l'ordonancement n'est pas totalement prévisible. En particulier l'étape représentée par l'objet E_1 peut intervenir à n'importe quel moment au cours du déroulement du calcul.

Naturellement qui dit ordre même partiel dit ordre tout de même, et ceci implique que certaines contraintes temporelles sont exprimées. Il s'agit des relations temporelles présentes dans l'ensemble $\mathscr{R}$ de la structure d'une part et d'autre part de relations implicites dues à l'organisation hiérarchique.

Ces dernières sont quelques peu particulières, car elles s'expriment au niveau de la structure elle-même et non pas au niveau supérieur.

Formulation 6.3 *Soit* S *une structure avec un ensemble d'objets enfants $\mathscr{E} = \{O_1, \ldots, O_n\}$, pour chaque objet enfant, on a les relations temporelles dans l'ensemble $\mathscr{R}$ de S :*

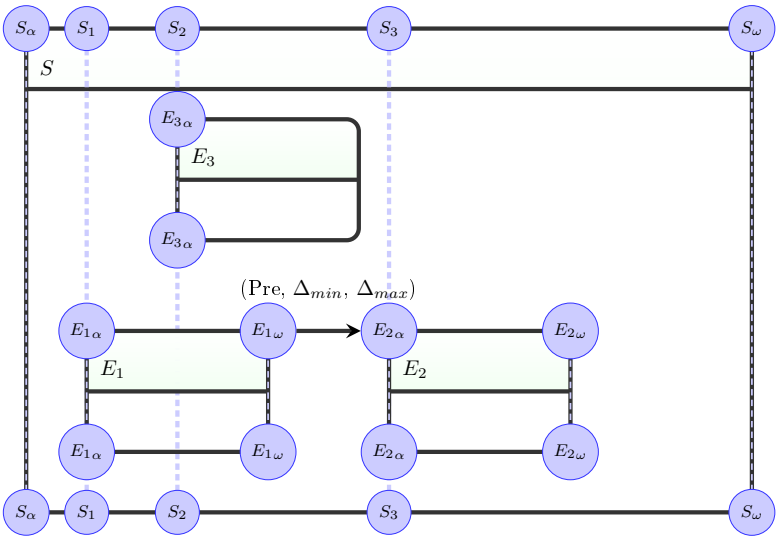
- $< pre, start(S), start(O_i), \Delta 1_{min}, \Delta 1_{max} >$
- *Si O_i est ponctuel :*

$$< pre, start(O_i), end(S), \Delta 2_{min}, \Delta 2_{max} >$$

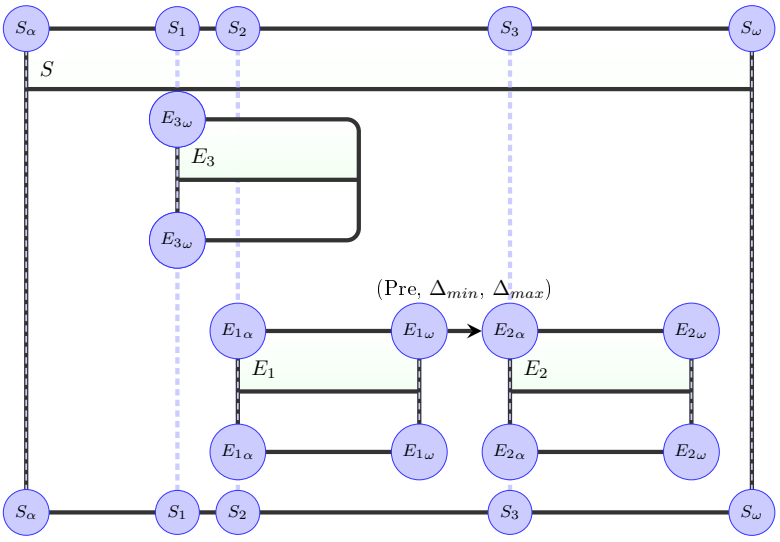
Si O_i est non-ponctuel :

$$< pre, end(O_i), end(S), \Delta 2_{min}, \Delta 2_{max} >$$

Ces relations traduisent simplement le fait que le déroulement d'un objet a lieu "pendant" celui de son objet parent. A priori, on pourrait penser que des relations ne sont guère utiles puisque l'origine du référentiel temporel d'une structure coïncide avec son début, mais vis à vis des niveaux hiérarchiques supérieurs, ces relations représentent une part importante de l'ordre partiel entre les points de contrôle de la structure.



(a) Configuration initiale de l'ensemble $\mathscr{P}$



(b) Une autre configuration de l'ensemble $\mathscr{P}$

FIG. 6.1 – Modification de l'ordre de l'ensemble des points de contrôle d'une structure

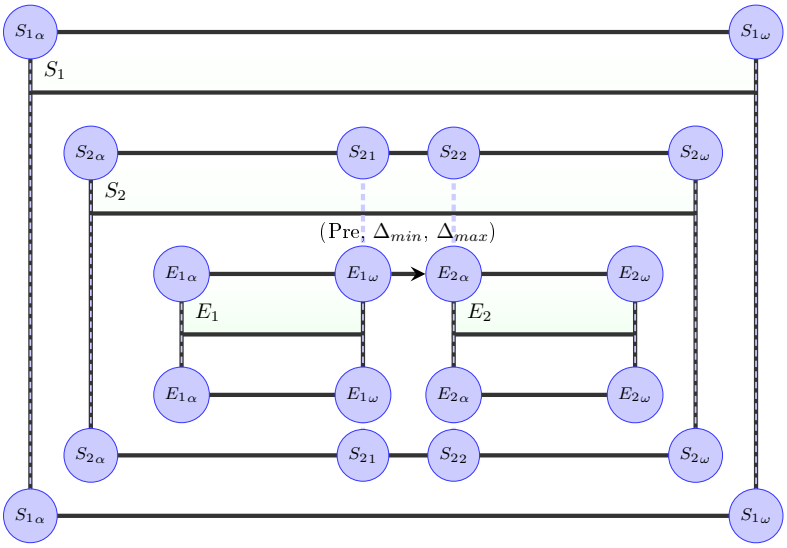


FIG. 6.2 – Un exemple de synthèse directe d'une relation temporelle $< Pre, \Delta_{min}, \Delta_{max} >$ depuis une structure vers le niveau hiérarchique supérieur (relation $< Pre, r(S_2).\Delta_{min}, r(S_2).\Delta_{max} >$).

Car en effet, placer des points de contrôle intermédiaires sur une structure, c'est aussi exporter dans les niveaux superieurs les relations temporelles exprimées entre les événements liés à ces points de contrôle ; exactement comme les processus propagent leurs contraintes intrinsèques.

Rappelons qu'un point de contrôle d'un structure à la même date vis à vis des niveaux supérieurs que l'événement qu'il représente :

$$date(pc, parent(S)) = date(control(pc), parent(S))$$

On a alors :

Propriété 6.1 *Soit S une structure qui n'est pas racine :*

$$\begin{array}{l} \forall pc_1, pc_2 \in \mathscr{P} \ t.q. \ control(p_1) = \sigma_1, control(p_2) = \sigma_2, \\ (\exists r =< type, \sigma_1, \sigma_2, \Delta_{min}, \Delta_{max} > \in \mathscr{R}(S)) \Rightarrow \\ \exists r' =< type, pc_1, pc_2, r(S).\Delta_{min}, r(S).\Delta_{max} > \in \mathscr{R}(parent(S)) \end{array}$$

Cette propriété est simplement un passage direct d'une relation temporelle d'une structure au niveau supérieur, moyennant un changement de quantum temporel dans l'expression des valeur Δ_{min} etΔ_{max}.

Un tel passage est illustré sur la figure 6.2. On y trouve une structure S_1 ayant pour enfant une structure S_2, ayant elle même pour enfant les objets simples E_1 et E_2. Les événements de S_2, correspondant à la fin de E_1 et au début de E_2 font l'objet de points de contrôle de E_2 (P_2 et P_3). De plus, il existe au niveau de S_2 une relation de précédence liant la fin de E_1 au début de E_2 :

$$\mathscr{R}(S_2) = \{< pre, end(E_1), start(E_2), \Delta_{min}, \Delta_{max} >\}$$

Il est assez clair qu'on peut déduire de cette situation une relation temporelle au niveau de S_1 :

$$r(S_2).\Delta_{min} \leq date(P_3, S_1) - date(P_2, S_1) \leq r(S_2).\Delta_{max}$$

Ce qui équivaut à l'existence dans $\mathscr{R}(S_2)$ d'une relation temporelle de précédence :

$$< pre, P_2, P_3, r(S_2).\Delta_{min}, r(S_2).\Delta_{max} >$$

Dans ce cas, la relation temporelle est une traduction directe d'une relation interne à la structure. Cependant, le panorama des formalismes pour raisonner temporellement nous a montré que des relations

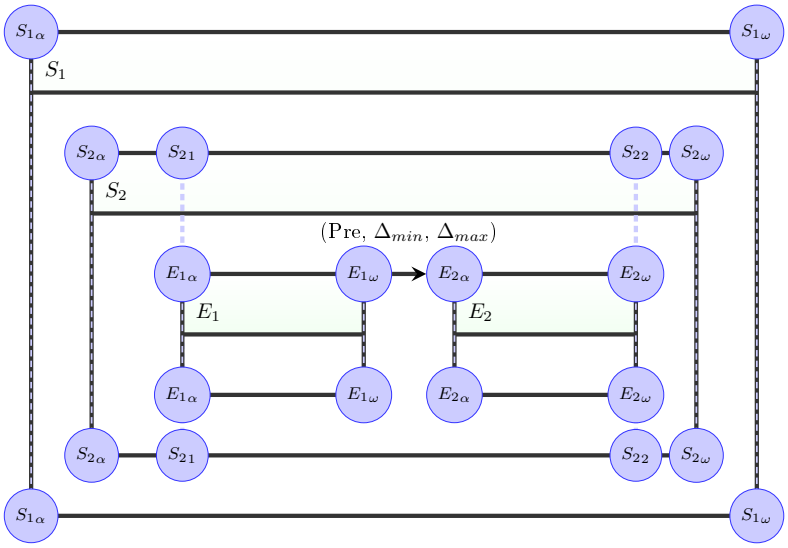


FIG. 6.3 – Un exemple de synthèse implicite de relations temporelles depuis une structure. Dans cet exemple, les objets E_1 et E_2 sont rigides, la relation $< Pre, \Delta_{min}, \Delta_{max} >$ de la structure S_1, implique donc une relation au niveau de S_1 <Pre, $r(S_2).(\Delta_{min} + \Delta_{E_1} + \Delta_{E_2})$, $r(S_2).(\Delta_{max} + \Delta_{E_1} + \Delta_{E_2})$>

temporelles entre des événements, induisaient des relations non écrites. Ce genre de situation se produit naturellement dans le cadre de notre modèle.

Pour s'en rendre compte reprenons l'exemple précédent, en changeant les événements associés à des points de contrôle en choisissant le début de E_1 et la fin de E_2 comme sur la figure 6.3, il n'existe plus de relation temporelle direct entre les événements associés aux points de contrôle P_1 et P_2 de S_2. Pourtant on peut déduire une relation temporelle entre ces points ; en effet, par transitivité sur les relations temporelles de "cohérence" des objets E_1 et E_2 et la relation entre la fin de E_2 et le début E_2, on obtient une relation entre P_1 et P_2.

Le mécanisme de répercution directe des relations temporelles d'une structure se trouve alors augmenté d'une propagation des relations temporelles indirectes.

En utilisant le symbol de déduction $\vdash$, on peut formaliser la propriété suivante :

Propriété 6.2 *Soit S une structure qui n'est pas racine :*

$$\begin{array}{l}\forall pc_1, pc_2 \in \mathscr{P}(S),\\ (\mathscr{R}(S) \vdash r =< type, control(pc_1), control(pc_2), \Delta_{min}, \Delta_{max} >) \Rightarrow\\ \exists r' =< type, pc_1, pc_2, r(S).\Delta_{min}, r(S).\Delta_{max} > \in \mathscr{R}(parent(S))\end{array}$$

Cette dernière propriété traduit bien la propagation "ascendante" (synthèse) des relations temporelles au travers de la hiérarchie d'une partition. Naturellement, elle n'a réellement de sens que si nous sommes capables de raisonner temporellement à partir des relations *pre* et *post*.

Cette dernière assertion soulève tout de même un léger problème qu'il nous faut éclairer. Dans la propriété 6.2, nous supposons que l'ensemble $\mathscr{R}$ permet de déduire uniquement des relations temporelles *Pre* ou *Post*, dont nous devons rappeler que les valeurs Δ_{min} et Δ_{max} sont positives. Comme une seule relation peut-être définie entre deux événements, certaines situations impliquent l'impossiblité de décrire la relation temporelle existant entre deux événements par une relation *Pre* ou *Post*. La figure 6.2 présente pareille configuration.

Sur cet exemple simple, les objets O_1 et O_2 sont définis comme rigides et entretiennent une relation de précédence sur leurs débuts avec des durées telles que :

$$\Delta_{0_2} + \Delta_{min} < \Delta_{O_1}$$

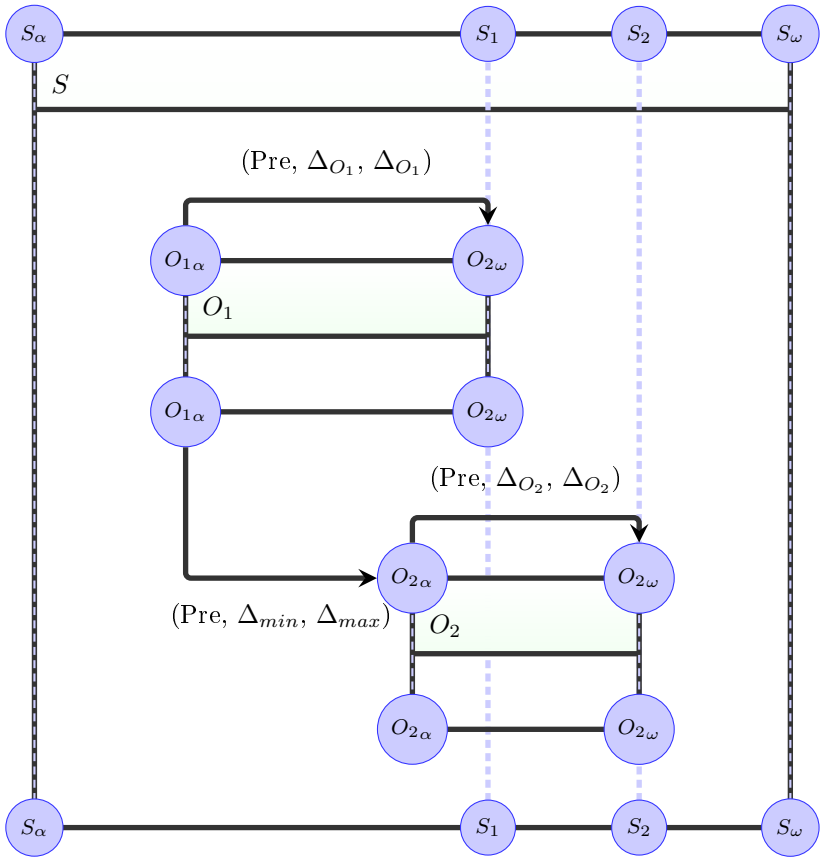


FIG. 6.4 – Un exemple de configuration dont la propagation des contraintes temporelles implique la création d'un point de contrôle. L'expression des contraintes temporelles entre $end(O_1)$ et $end(O_2)$ grace à des relations *Pre* et *Post* nécessite l'instanciation du point de contrôle P_5. Cette instanciation est ici indépendante de la volonté de l'utilisateur.

et

$$\Delta_{O_1} < \Delta_{0_2} + \Delta_{max}$$

En outre, nous supposons que le compositeur a choisi de ne voir représentés comme points de contrôle intermédiaires que $end(O_1)$ et $end(O_2)$.

Par conséquent, la relation temporelle entre $end(0_1)$ et $end(O_2)$ est la suivante :

$$\Delta_{min} + \Delta_{O_2} - \Delta_{O_1} \leq date(end(O_1), S) - date(end(O_2, S) \leq \Delta_{max} + \Delta_{O_2} - \Delta_{O_1}$$

Or les deux bornes n'étant du pas du même signe, il est impossible de traduire cette propriété directement avec une relation *Pre* ou *Post*.

Pour palier ce problème, il est possible d'utiliser l'événement $start(O_1)$ puisque la contrainte temporelle entre $end(O_1)$ et $end(O_2)$ est équivalente à la conjonction de relations temporelles suivantes :

$$\begin{array}{c} < pre, start(O_1), end(O_1), \Delta_{O_1}, \Delta_{O_1} > \\ \wedge \\ < pre, start(O_1), end(O_2), \Delta_{O_2} + \Delta_{min}, \Delta_{O_2} + \Delta_{max} > \end{array}$$

Par conséquent, pour représenter la contraintes sur l'intervalle séparant $end(O_1)$ et $end(O_2)$, à l'aide de relation *Pre* ou *Post*, il faut faire intervenir l'événement $start(O_1)$, et pour pouvoir propager cette contrainte, il est nécessaire de créer "automatiquement" un point de contrôle intermédaire lié à cet événement. Ce point de contrôle ne procède pas d'une volonté du compositeur mais découle du choix des relations *Pre* et *Post* comme écriture du temps dans les partitions. En ce sens, il est partie intégrante de la temporalité implicite de la structure S.

Finalement, la propriété traduisant la propagation des contraintes temporelles est la suivante :

Propriété 6.3

$$\begin{array}{l} \forall pc_1, pc_2 \in \mathscr{P}(S),\ \mathscr{R}(S) \vdash Rel(pc_1, pc_2)\ avec : \\ Rel(pc_1, pc_2) \Leftrightarrow \{(< type, control(pc_{1_i}), control(pc_{2_i}), \Delta_{i_{min}}, \Delta_{i_{max}} >)\}_i \\ \qquad \Rightarrow \\ \qquad\qquad \{pc_{i_1}, pc_{i_2}\}_i \subset \mathscr{P}(S) \\ \qquad\qquad\qquad \wedge \\ \{< type, pc_{1_i}, pc_{2_i}, r(S).\Delta_{i_{min}}, r(S).\Delta_{i_{max}} >\}_i \subset \mathscr{R}(parent(S)) \end{array}$$

Il s'agit ici d'analyser l'ensemble de relations $\mathscr{R}(S)$ de la structure S pour en déduire les relations temporelles implicites qu'il implique, exprimées sour la forme de relations *Pre* et *Post*.

Nous passons volontairement sous silence les méthodes et structures qui permettraient dans le cas général de détecter le ou les événements à expliciter pour propager les contraintes. Nous aborderons cette question au travers de reflexions autour de l'outil permettant la manipulation des partitions.

6.3 Relations implicites et branchements

La relative simplicité de la définition d'un branchement ne doit pas nous masquer leurs implications temporelles souvent majeures. L'ensemble de contraintes temporelles Γ_t d'un processus ne contient d'ailleurs pas uniquement des contraintes entre les étapes de calcul du processus lui-même, mais également des contraintes avec les étapes de calcul de processus avec lesquels il entrediendrait des branchements.

Pour illustrer ce type de situations dans un cadre de composition, développons l'exemple du processus de traitement sonore effectuant l'inversion d'un son enregistré en direct lors du déroulement de la partition. La figure 6.5 propose une configuration impliquant un tel processus.

Dans cet exemple, les objets temporels ont le sens suivant :

- O_{cap} effectue une captation d'un signal sonore pendant le déroulement de la partition
- O_{inv} applique l'inversion d'un signal fourni en entrée
- O_{amp} procède à une amplification de son flux d'entrée pour l'adapter au système d'écoute

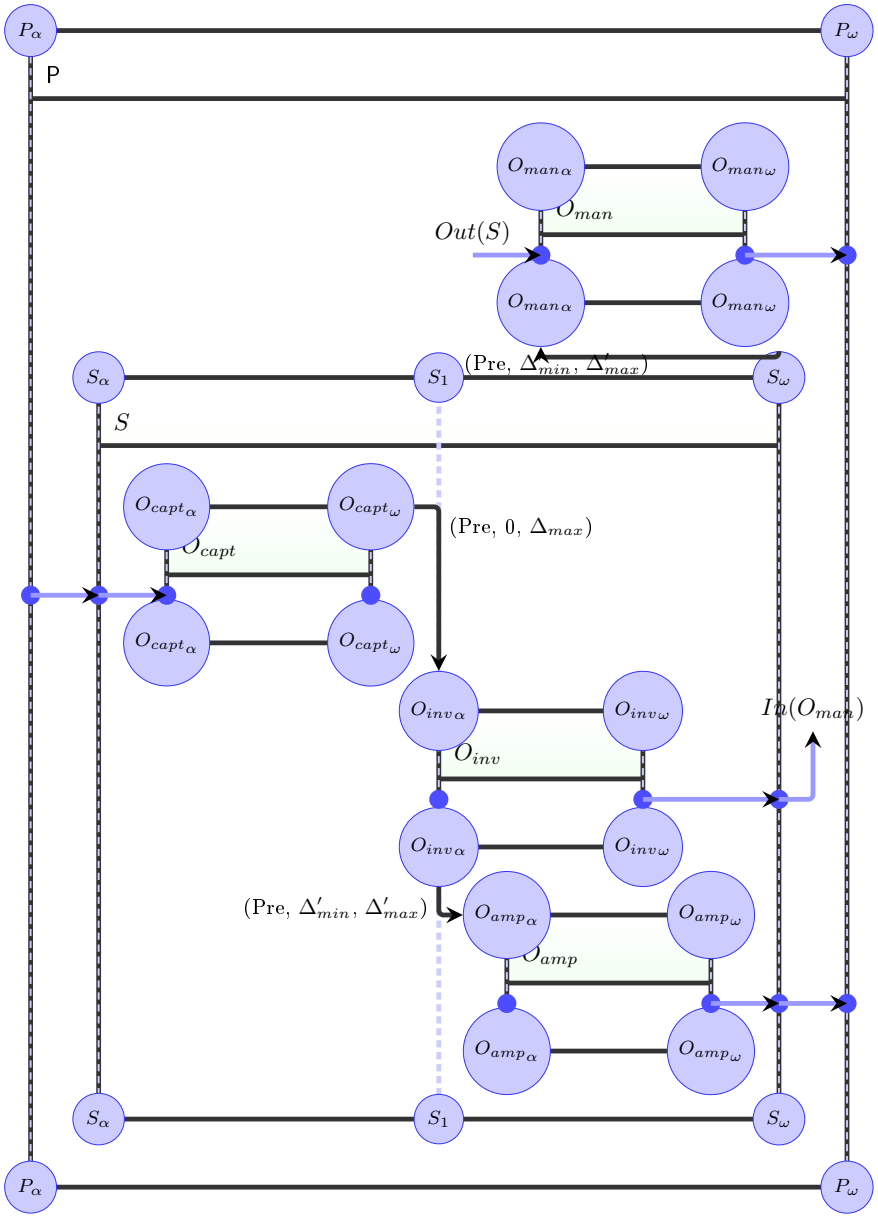


FIG. 6.5 – Un exemple de branchements impliquant des contraintes temporelles. Un processus de captation O_{capt} fournit un signal exterieur à un processus d'effet d'inversion O_{inv}. Ce dernier processus envoie le résultat de son calcul à un processus d'amplification O_{amp} et vers un porcessus de traitement O_{trait} situé au niveau hiérarchique supérieur. Le processus O_{amp} lui-même communique son résultat dans les niveaux supérieurs. Les caractéristiques des processus en question implique des relations temporelles entre les objets temporels les représentant. La logique du processus d'inversion implique la relation entre $end(O_{capt}$ et $start(O_{inv})$, tandis que les temps de calcul du processus d'inversion se traduit par les relations entre $start(O_{inv})$ et $start(O_{amp})$ et entre $start(O_{inv})$ et $start(O_{trait})$.

Plusieurs relations temporelles découlent alors du réseau de branchements entre les objets temporels. Le branchement de la sortie de l'objet O_{cap} fournissant le flux audio à l'entrée de l'objet O_{inv} effectuant l'inversion, implique immédiatement une relation de précédence entre $end(O_{cap})$ et $start(O_{inv})$ pour des raisons logiques. Cette relation doit pouvoir se déduire d'une contrainte de l'ensemble Γ_t du processus d'inversion.

Supposons à présent, que l'on souhaite que le résultat de cette inversion soit diffusé en temps réel. La sortie de l'objet O_{inv} doit être branchée sur l'entrée d'un objet O_{amp} pour la diffusion. Cette diffusion ne pouvant débuter avant la production des résultats de l'inversion, cela implique une relation de précédence entre $start(O_{inv})$ et $start(O_{amp})$. En outre, l'inversion nécessite naturellement un temps de calcul avant de fournir ses premiers résultats, faible pour une opération simple comme une inversion, il n'est cependant pas négligeable au vue des fréquences d'échantillonage courrantes pour les processus sonores. Par conséquent, la relation de précédence sus-citée possède une valeur minimum d'intervalle non nulle.

Le flux de sortie de O_{amp} est dirigé vers la sortie de S en vue d'être diffusé au travers de la sortie de P. Supposons maintenant, que le compositieur souhaite en plus disposer du son inversé pour effectuer un autre traitement pour diffusion. Pour des raisons d'organisation (certes obscures sur cet exemple simple, mais tout à fait envisageables pour des cas plus élaborés), le compositeur place l'objet effectuant ce nouveau traitement au même niveau hiérarchique que S, ce qui signifie que le branchement de la sortie de O_{inv} vers O_{trait} s'effectue par l'intermédiaire des sorties de S. De la même manière qu'il existe une relation de précedence entre $start(O_{inv})$ et $start(O_{amp})$, il doit exister la même relation entre $start(O_{inv})$ et $start(O_{trait})$. Cette relation ne peut s'exprimer qu'au travers d'un point de contrôle (pc_2) de S puisque les objets sont à des niveaux hiérachiques différents. Tout comme la relation entre $start(O_{inv})$ et $start(O_{amp})$, la spécification de ce point de contrôle et de la relation temporelle associée, ne sont pas l'expression d'une volonté de composition, mais les conséquences de relations fonctionnelles entre les processus des objets.

Cet exemple met en lumière les différentes implications temporelles des branchements au sein des structures. Dans cette présentation, nous supposons que l'ensemble de contraintes intrinsèques Γ_t d'un processus contient la description de ces relations. Comme pour les contraintes quantitatives entre les étapes de calcul du processus, cette possibilité s'appuie sur notre capacité à décrire les processus suffisamment précisément pour déduire leurs propriétés intrinsèques.

Cependant, il est important de noter que comme dans le cas des structures, la transposition des contraintes temporelles dans les niveaux hiérarchiques supérieurs peut impliquer l'intervention d'événements en dehors de la volonté directe du compositeur. Ce phénomène provient du fait que notre formalisme ne permet pas seulement d'écrire la musique mais également les calculs algorithmiques qu'elle implique dans sa réalisation. Or ces calculs procèdent d'une écriture et d'une logique qui ne sont pas nécessairement celles de la composition musicale. Ainsi dans l'exemple de la figure 6.5, l'exhibition du point de contrôle p_2 de S et des relations temporelles héritées n'ont qu'un rapport assez lointain avec le travail du compositeur sauf à considérer qu'il produit lui-même les processus qu'il emploie, en prenant soin que les contraintes intrinsèques qu'il introduit dans ce travail algorithmique fassent sens musicalement. Sans l'exclure totalement, ce type de pratique ne constitue pas la majorité des cas.

La question d'un formalisme de description des processus prend alors une importance cruciale, car un système utilisant notre formalisme de partitions devrait être capable de s'appuyer sur ce formalisme pour déduire systématiquement les propriétés de ces derniers. Le panorama de la temporalité implicite des objets musicaux des partitions, plaide pour un formalisme complet mélant une approche purement temporelle et une approche *processus concurrents*.

Remarque 6.1 *Nous l'avons déja signalé, le dialogue entre la temporalité intrinsèque des objets musicaux et celle de la partition est une part importante du processus de composition. Imposer systématiquement un respect strict des contraintes des objets ou simplement les exhiber, oriente les conditions de ce dialogue. Interroger les limites du matériau musical utilisé, fleurter avec elles pour obtenir des résultats inattendus sont des pratiques répandues. Sécuriser totalement et systématiquement le rendu musical d'une composition peut s'avérer alors incomptatible avec l'approche de certains compositeurs. Cette remarque aurait pu prendre place dans le chapitre concernant l'outil de manipulation des partitions, celui-ci pouvant s'adapter avec l'approche du compositeur en atténuant plus ou moins la représentation des contraintes intrinsèques.*

Cependant, ces mécanismes étant au cœur de notre formalisme, il est probable que celui-ci ne s'accorde pas avec la pratique de certains compositeurs.

Chapitre 7

Un modèle de partitions interactives

Une fois le matériau musical représenté au travers des partitions statiques, il est possible de définir explicitement des modalités d'interaction avec lui. Comme nous envisageons l'interaction par des contrôles discrets pilotant le déclenchement d'événements, son écriture s'effectue par l'introduction de points d'interaction.

De plus, la réaction de la partition à des modifications au cours de son exécution est une caractéristique propre à chaque pièce que le compositeur peut préciser à l'aide de contraintes temporelles globales.

D'un point de vue plus théorique, nous nous intéressons dans ce chapitre à la possibilité de modifier en temps réel les valeurs des variables de la partition statique. Comme nous nous restreignons à l'interprétation par modifications agogiques, seules les variables temporelles sont modifiables en temps réel. Cette possibilité revient à perturber le système constitué des variables temporelles et des contraintes qui les lient. Il est donc important de distinguer les valeurs de variables écrites dans la partition statique, des valeurs que prendront ces variables pendant l'exécution.

Dans ces conditions, les stratégies de réaction aux modifications introduites à l'exécution consistent à des propagations des modifications s'appuyant sur les méthodes associées aux contraintes.

7.1 Définition

Définition 7.1 *Une partition interactive Pa_{int} se définit comme un couple :*

$$Pa_{int} =< Pa_{stat}, \mathscr{I} >$$

dans lequel :

- *Pa_{stat} est une partition statique*
- *$\mathscr{I}$ est un ensemble de points d'interaction*

7.1.1 Les points d'interaction

Un point d'interaction permet de faire de lien entre un contrôle discret et le déclenchement d'un événement de la partition.

Définition 7.2 *Soit $Pa_{int} =< Pa_{stat}, \mathscr{I} >$ une partition interactive, un point d'interaction* pi *de l'ensemble $\mathscr{I}$ se définit comme un couple :*

$$pi =< \sigma, p >$$

avec :

σ *: un événement de Pa_{stat}*

p *: un processus qui consiste en l'écoute de l'environnement exterieur et l'attente de la réception d'un message spécifique*

Un événement auquel est associé un point d'interaction est appelé *événement dynamique*, dans le cas contraire, il est dit *statique*.

On peut exprimer la relation entre un événement dynamique et le point d'interaction qui le contrôle par un propriété temporelle :

Propriété 7.1 *Soient une partition interactive* $Pa_{int} =< Pa_{stat}, \mathscr{I} >$ *et* $pi =< \sigma, p >$*, en étendant la fonction date,* **pendant l'exécution** *on a :*

$$date(\sigma, abs) = date(pi, abs)$$

Un point d'interaction étant un événement extérieur la fonction *date* ne peut être définie pour lui que relativement au référentiel absolu.

Naturellement, l'intervention d'un point d'interaction consitue un choix de valeur pour la date absolue de l'événement qu'il déclenche. Ce choix de valeur implique des modifications éventuelles au sein de la partition, si la valeur imposée par le point d'interaction ne correspond pas à celle qui est écrite. La nature de ces modifications peut s'exprimer par la description d'une contrainte liant la date absolue d'un événement aux autres variables de la partition. Comme les autres contraintes que nous avons introduites, elle dispose de méthodes de propagation parmi lesquelles le compositeur va être amené à choisir afin d'orienter la modification des autres variables.

Définition 7.3 *Pour chaque événement* σ *d'une partition, on définit une contrainte* c_σ *telle que :*
- V *:* $\{r(parent(\sigma), \ldots, r(racine), date(parent(\sigma), abs), date(\sigma, abs), date(\sigma, parent(\sigma))\}$
- r *:* $date(\sigma, abs) = date(parent(\sigma), abs) + \frac{1}{r(parent(\sigma)) \times \ldots \times r(abs)}.date(\sigma, parent(\sigma))$
- $\mathscr{M}$ *:*
  - m_1 *:* $date(\sigma, abs) \leftarrow date(\sigma, abs)'$
  - m_2 *:* $date(\sigma, S) \leftarrow date(\sigma, S)'$
  - m_3 *:* $r(abs) \leftarrow r(abs)'$
  - $\ldots$
  - m_n *:* $r(parent(\sigma)) \leftarrow r(parent(\sigma))'$

  Les nouvelles valeurs affectées aux variables sont des applications directes de la relation.

Cette contrainte correspond exactement au calcul récursif de la date absolue d'un événement à partir de la date de cet événement dans le référentiel de sa structure parent. On peut la considérer comme implicite, voire redondante par rapport à la définition des partitions statiques. Cependant elle permet de formaliser la réaction d'une partition au déclenchement d'un point d'interaction de la même manière que les modifications des valeurs de variables pour les partitions statiques : par le choix d'une méthode de propagation.

Les méthodes proposées avec la contrainte s'interpètent assez simplement d'un point de vue musical. Notons que la méthode m_1 concerne les événements statiques, tandis que les méthodes m_2 à m_n sont possibles pour les événements dynamiques.

Les méthodes

- m_1 : cette méthode est celle choisie systématiquement pour un événement statique, la date absolue de l'événement est alors calculée depuis la date de l'événement dans la structure parent, en fonction des valeurs courantes de "tempo" aux différents niveaux hiérarchiques
- m_2 : le choix de cette méthode implique la répercussion directe sur la date de l'événement de la date absolue imposée par le déclenchement du point d'interaction. Cette modification de la date de l'événement va éventuellement entrainer la solicitation des méthodes de propagation des contraintes d'intervalles.
- m_3 à m_n : ces méthodes traduisent la modification de la date absolue de l'événement comme un changement de tempo de l'une des structures constituant le chemin de puis la racine jusqu'à l'événement.

L'utilisation de cette contrainte s'appuie sur l'existence de relation temporelle de précédence entre la date de début de la structure parent et la date de l'événement. Cette relation est assurée par les contraintes de cohérence d'une structure.

Concernant les choix du compositeur vis à vis des points d'interaction, outre celui de la méthode propagation de la contrainte précédente, il peut bien sûr définir le processus d'écoute en attente du message de déclenchement. Cependant ce choix n'est pas totalement libre. En effet, choisir des messages semblables équivaut dans certain cas à induire des relations de synchronisation entre les événements interactifs. Nous chercherons systématiquement à éviter ce genre de situation pour circonscrire l'écriture du temps aux relations temporelles et aux contraintes globales.

7.2 Stratégies d'adaptation

Comme nous l'avons vu avec la contrainte précédente, il existe deux approches pour réagir au déclenchement d'un point d'interaction : modifier un tempo ou modifier une date. Le premier cas permet des modifications globales pouvant impacter un grand nombre d'événement, tandis que le second utilise les contraintes locales pour modifier un nombre limité de variables temporelles.

7.2.1 Changements de tempo

La modification du tempo est une opération très courrante dans la musique rythmée. L'intérêt d'utiliser des contraintes temporelles pour effectuer cette opération dans les partitions interactives, est de contrôler cette modification par le déclenchement d'un seul point d'interaction. L'anticipation du déclenchement par rapport à la date attendue va conduire à une augmentation de tempo tandis qu'un délais impliquera un ralentissement du tempo.

La figure 7.1 présente un exemple très simple de modification de tempo par le déclenchement d'un point d'interaction.

En outre, il est possible de choisir le niveau hiérachique dont le tempo est modifié par la sélection d'une méthode de propagation pour la contrainte 7.3. Cette possibilité épouse bien des structures hiérarchiques rencontrées en musique. La figure 7.2 présente deux mesures du morceau *Blue Bossa*[16] et leur transcription dans le formalisme des partitions interactives.

Dans cet exemple, les parties correspondant aux deux mains du pianiste ont été placées chacune dans une structure associée : *mélodie* pour la main droite et *basse* pour la main gauche (celle-ci jouant les accords). Chacune dispose d'un rapport de quantum spécifique avec la structure globale les incluant. Dans cette configuration, il est aisé de contrôler au travers d'un point d'interaction placé sur un événement de la mélodie, le tempo de la mélodie seule ou le tempo global.

Modification de tempo pour un comportement de type *rubato*

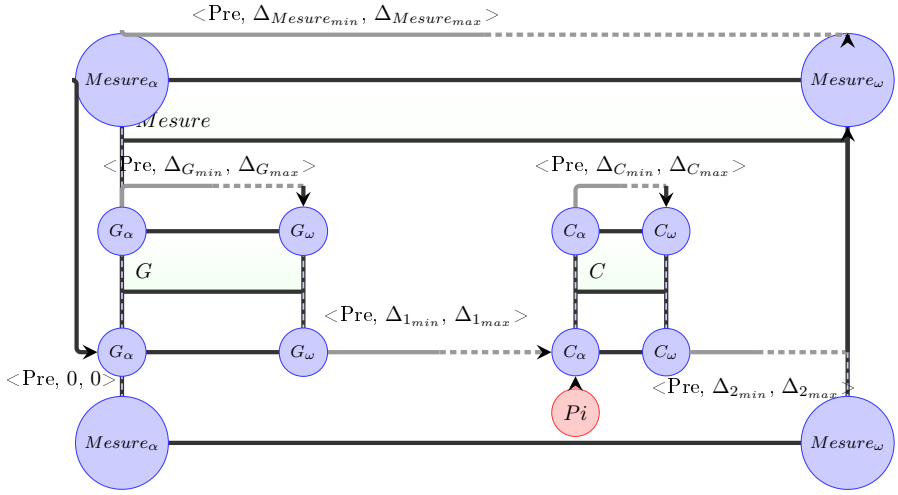
Le *rubato* est une variation temporelle autour d'un tempo donnée de telle manière qu'une accélaration (resp. une descélération) dans une première partie corresponde une descélération (resp. accélération) dans une seconde partie. Pour obtenir ce type de réaction dans une partition, il est possible de lier les vitesses des deux parties du *rubato* au travers d'une combinaison linéaire.

Nous proposons un exemple basé sur une mesure du *Nocturne en Mi mineur*[17] de Chopin. Les interprétations des pièces de l'époque romantique usent très souvent du *rubato* pour insufler à la pièce un mouvement ondulant.

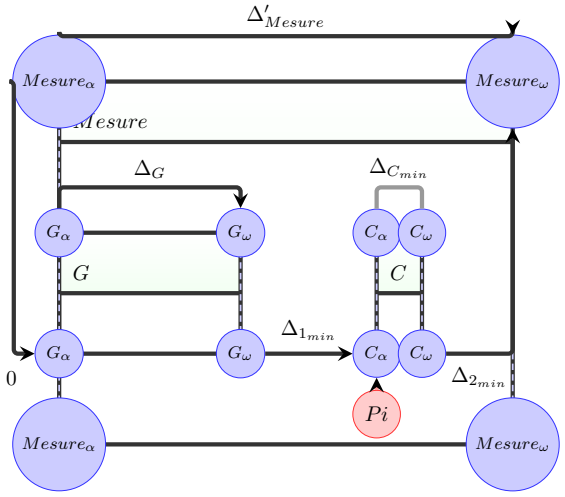
La figure 7.3(a) présente la mesure du *Nocturne* tandis que deux représentations de celle-ci sont données par la figure 7.3(b). Dans les deux cas, nous supposons que les sections du *rubato* sont les deux premiers et deux derniers temps de la mesure.

Dans la transcription de la figures 7.3(b), la partie basse est séparée en deux structures $basse_1$ et $basse_2$, avec leur ration associé. Ces deux ratios sont liés par une contrainte simulant la relation de

[16] Blue Bossa - Kenny Dohram 1965
[17] Nocturne en Mi mineur Opus 72 n°1 - Frédéric Chopin

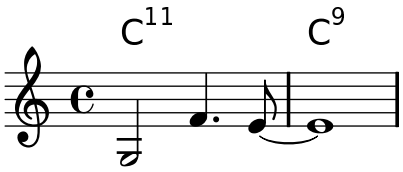


(a) Situation initiale

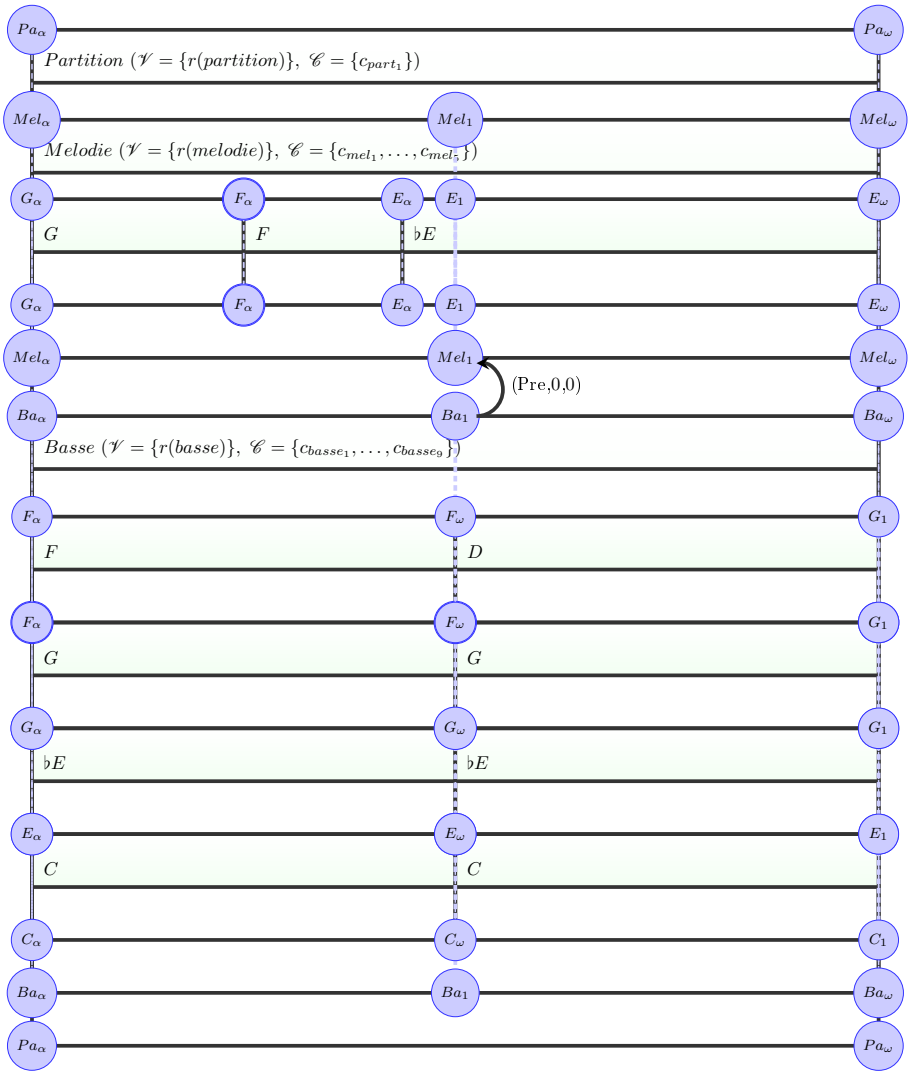


(b) Doublement du tempo par un déclenchement anticipé de P_i

FIG. 7.1 – Un exemple de changement de tempo suite au déclenchement d'un point d'interaction. Le point d'interaction Pi est déclenché au plus tôt, les variables Δ_1, Δ_C et Δ_2 sont alors réduits proportionnellement en fonction du nouveau tempo imposé par la modification de la valeur de Δ_1.

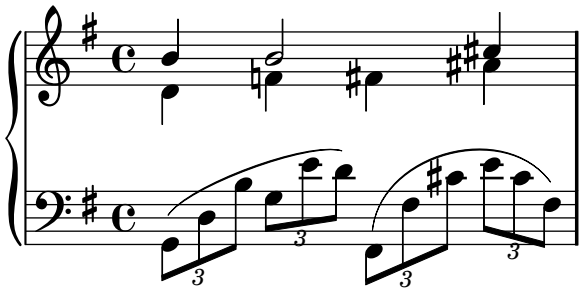


(a) Deux mesures de Blue Bossa

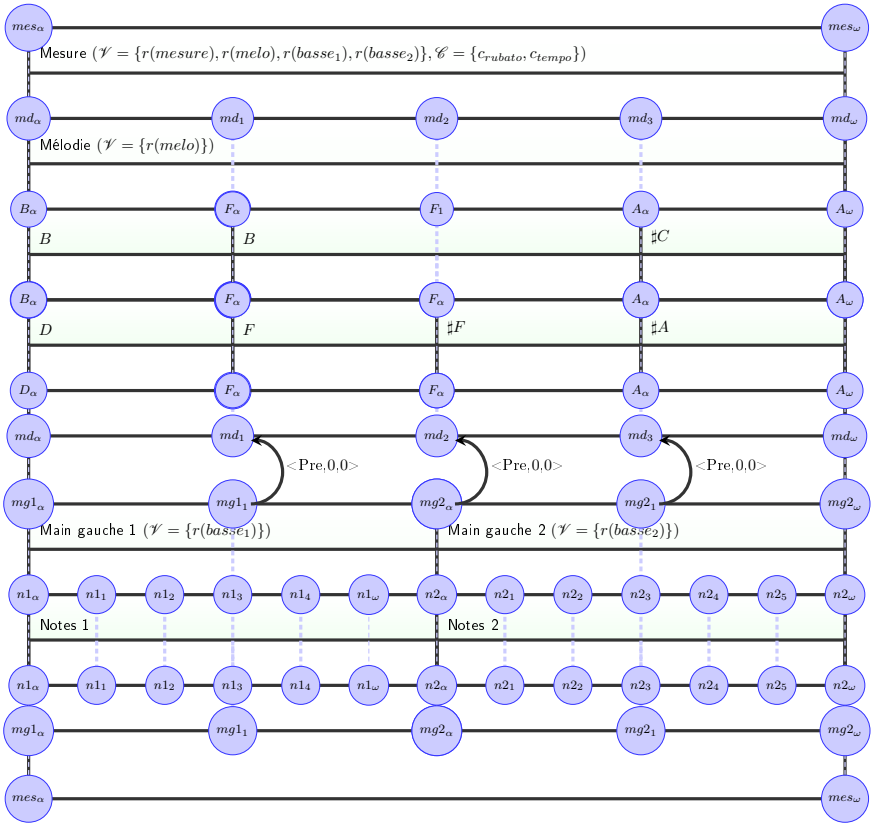


(b) Une représentation des deux mesures à l'aide d'une structure globale et deux sous-structures. Les relations de synchronisation ne sont pas toutes représentées, mais *Mélodie* et *Basse* synchronisent leurs début et fin avec ceux de *Partition*, tandis qu'à l'intérieur des sous-structures, les objets correspondant aux notes synchronisent leur fin avec le début du suivant, les premiers (resp. derniers) objets synchronisant leur début (resp. leur fin) avec celui de leur structure parent

FIG. 7.2 – Un exemple de transcription de 2 mesures d'une pièce instrumentale dans le formalisme des partitions interactives. La structure générale (*Partition*) contient elle-même 2 structures enfant (*Mélodie* et *Basse*) correspondant à chacune des mains du pianiste. Celles-ci étant liées au niveau de la fin de mesure par une relation temporelle pour synchroniser le changement d'accord sur la mélodie. Dans cet exemple la définition d'un point d'interaction sur un événements de la mélodie permet de modifier le tempo de la mélodie uniquement ou de la partition globale, selon la méthode de propagation choisie



(a) Une mesure du *Nocturne*



(b) Une représentation de la mesure du *Nocturne*. Par consision, certaines relations de synchronisation ne sont pas représentées. La coïncidence de points de contrôle de différents objets indique une relation de synchronisation entre ces points. De plus on trouve des relations de synchronisation entre les débuts et fins des sutructures et ceux de leru premier et dernier enfants. Les objets *Notes* sont supposés représentés des processus capables d'effectuer les synthèses successives des notes de la basse.

FIG. 7.3 – Un exemple de comportement de type rubato sur une mesure du *Nocturne en Mi mineur Op.72* de Frédéric Chopin. Dans cette représentation, la mélodie est séparée de la basse, elle même représentée par deux sous-structures disposant de ratios spécifiques. Ces deux ratios $r(basse_1)$ et $r(basse_2)$ sont des variables de la structure *Mesure*, la contrainte c_{rubato} impose la relation entre ces deux variables assurant le rubato. Une modification sur $r(partie_1)$ va conduire à calculer une valeur pour $r(partie_2)$ qui sera utilisée à l'exécution de la seconde structure. Les contraintes c_{tempo_1} et c_{tempo_2} vont imposer les égalités successives de $r(melo)$ avec $r(basse_1)$ et $r(basse_2)$ pour accorder le tempo de la mélodie avec celui de la basse.

rubato, c_{rubato} qui impose l'égalité :

$$r(basse_1) + r(basse_2) = 2$$

où la valeur 2 traduit le fait qu'en dehors de toute modification, les tempo des deux parties sont égaux au tempo de la mesure, c'est à dire :

$$r(basse_1) = 1$$

$$r(basse_2) = 1$$

La méthode de propagation choisie pour la contrainte c_{rubato} permet le calcul de $r(S_2)$ en fonction de $r(S_1)$. Ce dispositif permet avec un point d'interaction placé sur l'un des événements de $basse_1$ en modifiant le tempo, d'obtenir un *rubato* sur la mesure sans nouvelles interventions du musicien.

Concernant la ligne mélodique, sa division en deux parties de la même manière que la basse n'est pas possible, la blanche couvrant les 2^{eme} et 3^{eme} temps de la mesure devant alors être séparée. Des subtilités quant à la représentation de la synthèse d'une même note sont peut-être envisageable, mais souhaitant rester dans un cas simple, nous avons opté pour la conservation d'un seul processus et donc objet pour cette note.

En conséquence la ligne mélodique est représentée par une unique structure, les changements de tempo de la basse devant se répercuter à la mélodie, nous ajoutons les contraintes $tempo_1$ et $tempo_2$ permettant la propagation des valeurs $r(basse_1)$ et $r(basse_2)$ lors de l'exécution des deux structures.

7.2.2 Modification de date

Le second type de réaction au déclenchement d'un point d'interaction est de répercuter directement le changement de la date absolue sur la date de l'événement interactif dans le référentiel de son objet parent. Les conséquences de cette modification de date peuvent être variées et dépendent en grande partie des relations temporelles de la structure parent de l'événement.

Nous proposons au compositeur de choisir parmi un ensemble de stratégies d'adaptation à pareille modification, celle qui souhaite voir utilisée dans chaque structure. Ces stratégies sont appelées *comportements d'intervalles*.

7.3 Comportements d'intervalles

Pour décrire la manière dont un changement de la date d'un événement va venir modifier la partition, nous définissons des *comportements d'intervalle* qui sont en fait des règles de propagation de perturbation de valeurs d'intervalles. Un comportement d'intervalle est une donnée globale qui est la même pour tous les intervalles d'une structure.

La spécification d'un comportement est intéressante dans le cas où une relation temporelle est définie entre deux événements qui ne sont pas directement successifs. Ainsi, soit une relation temporelle rt, entre deux événements d'une structure S :

$$rt =< t, \sigma_1, \sigma_n, \Delta_{min}, \Delta_{max} >$$

Telle que :

$$\exists(\sigma_2, \ldots, \sigma_{n-1}) \text{ et } (rt_1, \ldots, rt_{n-1})$$

$$tel\ que\ \forall i \in [1, n-1],\ rt_i =< pre, \sigma_i, \sigma_{i+1}, \Delta_{i_{min}}, \Delta_{i_{max}} >$$

En notant Δ la valeur de l'intervalle entre σ_1 et σ_n i.e :

$$\Delta = date(\sigma_n, S) - date(\sigma_1, S)$$

et de la même manière :

$$\forall i \in [1, n-1],\ \Delta_i = date(\sigma_{i+1}, S) - date(\sigma_i, S)$$

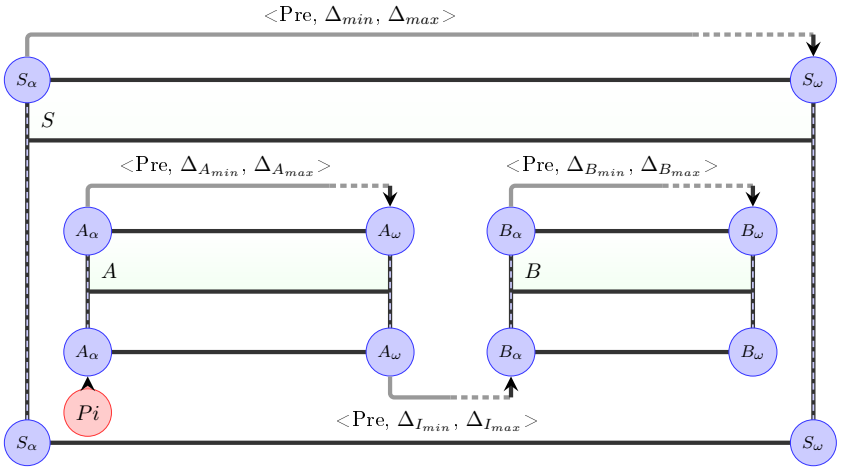


FIG. 7.4 – Une situation de définition d'un comportement d'intervalle. Pour ne pas surcharger la figure, les intervalles Δ_1 et Δ_2 séparant respectivement $start(S)$ et $start(A)$, et $end(B)$ et $end(S)$ ne sont pas représentés.

On a :

$$\Delta = \sum_{i=1}^{n-1} \Delta_i \tag{7.1}$$

Nous considérons cette égalité comme la relation imposée par une contrainte d'intervalles (combinaison linéaire sur des valeurs d'intervalles). Cette contrainte de même que celle proposée pour décrire la conséquence du déclenchement d'un point d'interaction est une contrainte implicite.

Comme précédemment, les comportements d'intervalle sont définis par des méthodes de propagation définies pour cette contrainte.

La relation temporelle rt entre les deux événements distants qui justifie la définition d'un comportement d'intervalle n'est pas nécessairement le reflet d'une volonté du compositeur. Précisément, les relations temporelles dites de cohérence, très nombreuses, sont le plus souvent le relais de comportement d'intervalle.

L'exemple certainement le plus simple d'une telle situation est présenté sur la figure 7.4.

Sur cet exemple, la relation rt en question est celle qui sépare le début d'une structure de sa fin. Par conséquent, l'intervalle Δ considéré est la durée de cette structure S. Les enfants A et B sont contraints par une relation de précédence. Les relations temporelles de cohérences entre le début de S et le début de A, ainsi qu'entre la fin de B et la fin de S, impliquent un ordre total entre les événements de S. La relation 7.1 lie donc la durée de S avec les intervalles séparant des événements successifs de S. En notant Δ_1 et Δ_2 les intervalles non représentés séparant $start(S)$ et $start(A)$ et $end(B)$ et $end(S)$, elle s'exprime par :

$$\Delta = \Delta_1 + \Delta_A + \Delta_I + \Delta_B + \Delta_2$$

On peut noter que cette contrainte est systématiquement identifiable pour une structure. Si il n'existe pas d'ordre total entre les événements, alors plusieurs contraintes de ce type sont décelables en fonction des relations temporelles entre les événements de S.

Il est important d'avoir à l'esprit que la relation 7.1 associée à S s'exprime dans le référentiel temporel S. Et qu'en particulier :

$$\Delta = date(end(S), S)$$

Et qu'il en va de même pour toute structure avec sa (ou ses) contrainte 7.1 associée.

Pour complêter l'exemple, un point d'interaction déclenche le début de A. Dans ces conditions, la modification de la date de début de A va impliquer une propagation au travers de la contrainte 7.1 et impacter les valeurs des autres intervalles.

Naturellement, plusieurs stratégies sont possibles pour effectuer ces modifications, c'est précisément la stratégie choisie pour les intervalles d'une structure que nous appelons le comportement d'intervalle de cette structure.

Modification de tempo et modification d'intervalles

Une propagation au travers des contraintes d'intervalles va induire des modifications des autres variables d'intervalles, mais avant de présenter les différents comportements, il nous faut mettre en lumière un cas particulier dans lequel, la propagation de la modification va induire des modifications de tempo au niveau des structures enfant.

Ainsi, en reprenant les notations de la figure 7.4, on peut supposer que dans cet exemple l'objet B (enfant de S) est lui même une structure. La variable Δ_B se trouve dans le membre droit de la contrainte d'intervalles de S (c_S) et donc à ce titre, elle peut être modifiée par un déclage du point d'interaction Pi.

En outre si B contient des objets enfant, alors la durée de B va être impliqué comme membre gauche d'une (ou plusieurs) contrainte d'intervalles (c_B), issues de l'ordre partiel entre les événements de B.

Cependant, les contraintes d'intervalles c_S et c_B ne s'expriment pas dans le même référentiel temporel, dans celui de S pour c_S et dans celui de B pour c_B.

Ainsi, Δ_B qui apparait dans c_S s'exrpime par :

$$\Delta_B = date(end(B), S) - date(start(B), S)$$

De plus, le membre gauche de c_B étant la durée B dans son propre référentiel temporel elle s'exprime simplement par $date(end(B), B)$.

Or par définition des ratio entre les quanta temporels :

$$date(end(B), B) = r(B).\Delta_B$$

Comme dans le cas du choix entre changement de tempo et changement de date en réaction au déclenchement d'un point d'interaction (section 7.1.1), une modification de Δ_B peut impliquer une modification de $r(B)$ ou de $date(end(B), B)$.

Dans le cas d'une modification de $r(B)$, l'égalité imposée par c_B reste valide et aucune modification ne sera induite sur les durées des objets enfant de B. Dans l'autre cas, ce genre de modifications sont susceptibles de se produire.

Ces deux approches correspondent à des choix d'écriture différents et nous proposons au compositeur de définir pour chaque structure lequel il préfère. Ce choix se fait pour une structure par rapport à sa structure parent, pour reprendre l'exemple précédent, c'est au niveau de B que le compositeur va pouvoir choisir entre la répercution de Δ_B sur $r(B)$ ou $date(end(B), B)$.

Il est important de noter que ce choix est indépendant du comportement d'intervalle défini pour la structure en question (B), et qui lui définit la stratégie de propagation au niveau de variables de durée des enfants de B.

7.3.1 Membre gauche/Membre droit

Les comportements d'intervalles disposent chacun de deux méthodes de propagation. Pour désigner ces deux méthodes, nous reprenons l'égalité 7.1 pour distinguer la méthode propageant une modification de Δ que nous appellerons *Modification du membre gauche*, et la méthode réagissant à une modification de l'un des Δ_i que nous appellerons *Modification du membre droit*.

Cette distinction est une conséquence directe de l'organisation hiérarchique de notre modèle. Pour reprendre l'exemple de la figure 7.4, si le déclenchement du point d'interaction intervient à une date différente de celle qui est attendue et il y a toutes les chances pour qu'il en soit ainsi, cette modification sollicitera la contrainte 7.1 de manière "interne". C'est-à-dire en propageant une perturbation sur l'un des Δ_i du membre droit de l'équation.

Ce faisant, il est tout à fait possible que Δ_B soit modifié. Or à supposer que B soit une structure, il aura alors une contrainte de type 7.1 associée. Mais dans cette équation, Δ_B sera le membre gauche de

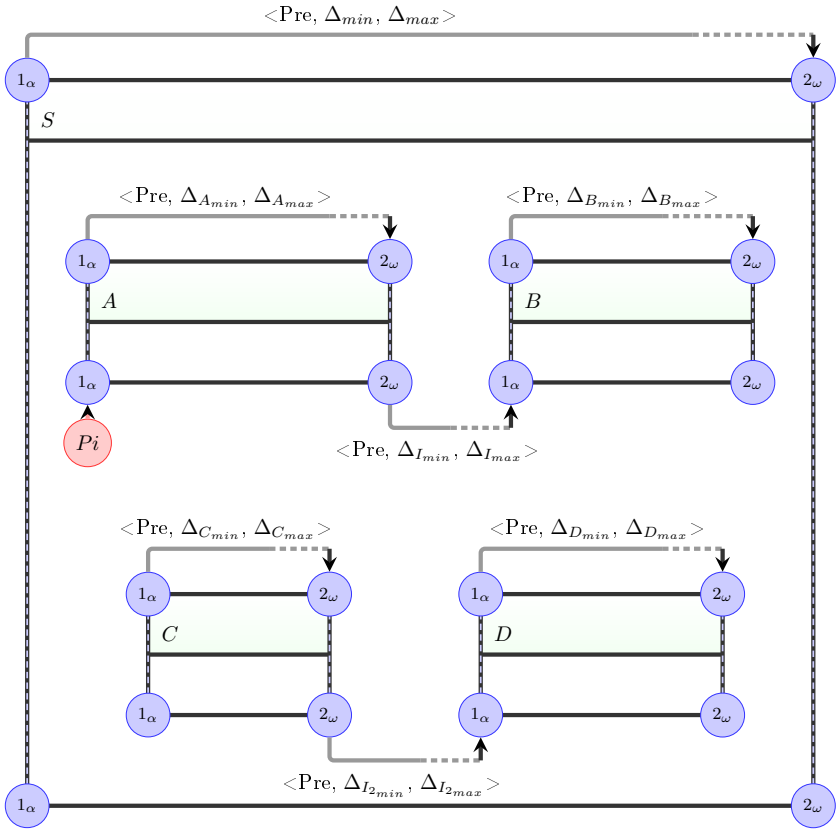


FIG. 7.5 – Une situation dans laquelle l'ordre partiel entre les événements d'une structure implique la définition de deux contraintes d'intervalle pour cette structure. Par concision, certains intervalles ne sont pre-présentés : Δ_1 sépare $start(S)$ et $start(A)$, Δ_2 sépare $end(B)$ et $end(S)$, Δ_3 sépare $start(S)$ et $start(C)$, et enfin Δ_4 sépare $end(D)$ et $end(S)$. On peut considérer que les listes d'intervalles $(\Delta_1\Delta_A\Delta_I\Delta_B\Delta_2)$ et $(\Delta_3\Delta_C\Delta_{I_2}\Delta_B\Delta_4)$ forment deux lignes de temps inédependantes devant se synchroniser en $end(S)$.

l'équation et il y aura alors une propagation vers les Δ_i au travers de la contrainte de cette perturbation que l'on pourra qualifier d'externe.

C'est pourquoi, il faut alors prévoir une méthode de propagation pour la pertubation de chacun des membres de l'équation ; mais que l'on se rassure, elles procèdent des mêmes mécanismes.

Un cas intéressant à signaler est celui pour lequel il existe plusieurs contraintes d'intervalles partageant un même membre gauche. Un exemple de cette situation très courante est proposé sur la figure 7.5. Comme nous l'avons signalé plus haut, à partir du moment où il n'existe pas d'ordre total entre les événements d'une structure, plusieurs contraintes d'intervalles cohabitent avec la durée de la structure comme membre droit. Sur l'exemple de la figure 7.5, en notant Δ_1, Δ_2, Δ_3 et Δ_4 les intervalles non représentés de la même manière que pour l'exemple précédent, on a :

$$\Delta = \Delta_1 + \Delta_A + \Delta_I + \Delta_B + \Delta_2 \tag{7.2}$$

$$\Delta = \Delta_3 + \Delta_C + \Delta_{I_2} + \Delta_D + \Delta_4 \tag{7.3}$$

Ces multiples contraintes traduisent les égalités entre les sommes d'intervalles menant du début à la fin de la structure. Dans ces conditions, la modification du membre droit de l'une de ces contraintes peut conduire à une modification de son membre gauche, et ainsi à la propagation de cette modification aux membres droits des autres contraintes d'intervalles de la structure. Comme c'est le cas avec le point

d'interaction sur la figure 7.5, qui modifiant Δ au travers de l'égalité 7.2, va propager cette modification aux variables de l'égalité 7.3.

On peut voir cette situation comme la conséquence d'une synchronisation de deux lignes de temps imposée au point $end(S)$. Cette synchronisation nécessitant des adaptations lorsque que la date de cet événement se trouve modifiée au cours de l'exécution. Les adaptations imposées dépendent du comportement choisi pour la structure.

Si on se souvient qu'il est question ici de faire de la propagation en temps réel, on peut entrevoir sur cet exemple que le temps qui s'écoule fixe progressivement la valeur des Δ rendant ainsi la modification de ceci impossible. Comme il n'existe qu'un ordre partiel entre les événements de la structure, il n'est possible de prévoir quelles variables seront encore modifiable au moment de la propagation de la modification induite par Pi. La formalisation des méthodes de propagation associées aux contraintes d'intervalles prend donc en compte cette notion, en calculant des nouvelles valeurs uniquement pour les intervalles dont l'évement origine n'a pas encore été déclenché (i.e. sa date n'a pas encore été instanciée).

Pour présenter les comportements, nous garderons l'exemple de la figure 7.4 comme situation initiale. Nous également une situation avec 2 contraintes comme dans l'exemple de la figure 7.5, pour donner un aperçu des conséquences de la propagation au travers de plusieurs contraintes.

Remarque 7.1 *Dans tous les exemples grahiques que nous fournissons en illustration des comportements d'intervalles, nous représentons les relations temporelles dont l'intervalle associé est modifiable avec une section en pointillés. Le lien entre la taille de section pointillé et les caractéristiques de la relation peut s'établir ainsi : la section pleine correspond à la taille minimum de l'intervalle, la section correspond à la différence entre cette valeur minimum et la valeur nominale (écrite dans la partition statique) de l'intervalle. Afin de ne pas surcharger les figures, nous ne dessinons pas de section pointillé représentant la différence avec la valeur maximale (celle-ci s'étenderait au delas de la valeur nominale). Lorsqu'une modification conduit à donner à un intervalle l'un de ses valeurs extrèmes, alors la relation est remplacée par cette valeur.*

7.3.2 Comportement "point d'orgue"

L'approche la plus simple est certainement celle qui consiste à réagir de la même manière lors de la présence d'un point d'orgue dans les partitions instrumentales, c'est-à-dire à répercuter sur la durée de l'intervalle globale la modification locale due au point d'interaction.

En notant Δ_i l'intervalle du membre droit de l'équation 7.1 finissant par l'événement interactif et δ_i la différence constatée par rapport à sa valeur attendue, la méthode de propagation du comportement "point d'orgue" pour une modification du membre droit s'exprime par :

– $m_1 : \Delta \leftarrow \Delta + \delta_i$

La propagation d'une modification sur le membre gauche va se répercuter sur le dernier intervalle chronologiquement parlant.

Ainsi, en notant δ la différence sur la valeur attendue de Δ et $Delta_j$ le dernier intervalle de la contrainte, la méthode de propagation d'une modification du membre gauche s'écrit :

– $m_2 : \Delta_j \leftarrow \Delta_j + \delta$

La figure 7.6 présente un cas de modification avec un comportement de point d'orgue, de la situation initiale de la figure 7.5. On peut y observer les deux méthodes de propagation. La modification de Δ_1 par le point d'interaction conduit à une modification de Δ. Cette modification de Δ se répercutant sur Δ_4, dernier intervalle dans l'ordre chronologique de la ligne de temps ($\Delta_3\Delta_C\Delta_{I_2}\Delta_B\Delta_4$).

Sur cet exemple, nous avons supposé que Δ a été étendu à son maximum (Δ_{max}) par le décalage du point d'interaction. Cette augmentation s'effectue par propagation depuis le membre droit dans l'égalité 7.2, et conduit à une modification de Δ_4 par propagation depuis le membre gauche à travers l'égalité 7.3, Δ_4 prenant alors sa valeur maximum $\Delta_{4_{max}}$.

7.3.3 Comportements particuliers

Notre modèle, outre adopter des comportements existants dans la musique instrumentale permet également de définir des comportements inédits dans ce contexte, et plutôt inspirés par des problèmes

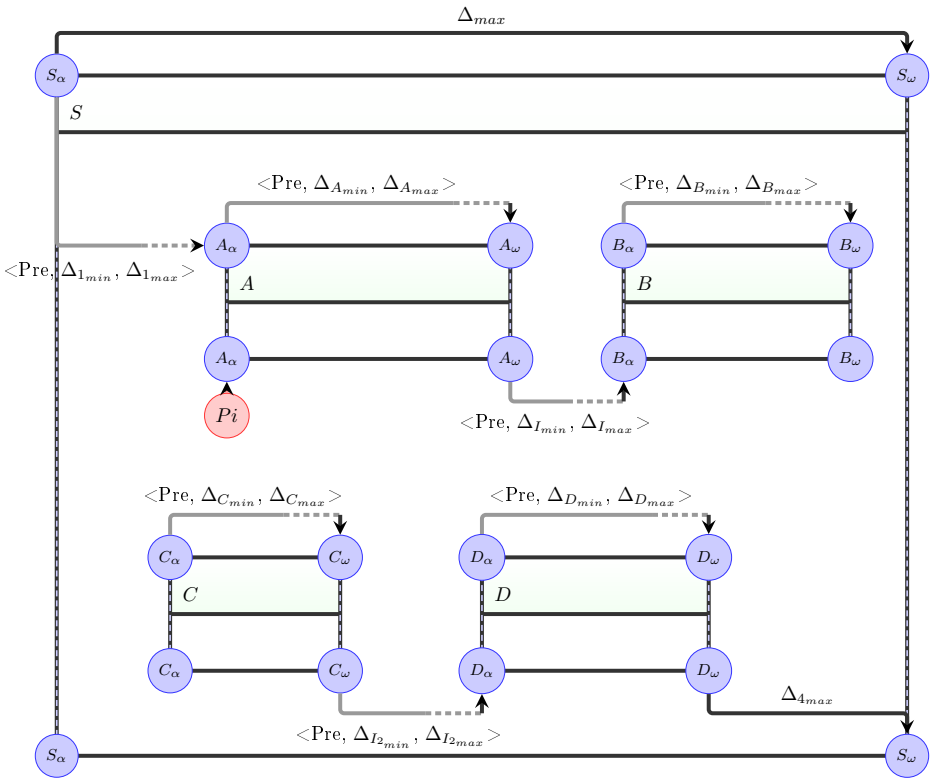


FIG. 7.6 – Une modification avec comportement point d'orgue de la situation initiale de la figure 7.5. La modification de Δ_1 induit un agrandissement de Δ qui se répecrute alors sur Δ_4, les autres variables restant inchangées. Nous supposons que la modification conduit à agrandir Δ et Δ_4 à leurs valeurs maximum.

d'ordonnancement plus généraux. Les variations sont naturellement infinies et nous avons choisi d'en mettre en avant seulement deux. Elles correspondent à des comportements simples à appréhender pour le compositeur et rencontrés dans certains arts numériques autres que la musique.

Nous désignons ces comportements par *écrasement chronologique* et *écrasement anti-chronologique*. Il s'agit dans les deux cas de propager la modification due au point d'interaction non plus sur une variable de la relation 7.1 mais sur plusieurs d'entre elles. Pour épargner au lecteur de lourdes formules, nous présentons ces comportements par des exemples graphiques. Les amateurs de formalisation pourront se reporter à l'annexe A pour satisfaire leur curiosité. Dans chacun des cas nous présentons les étapes successives de propagation au travers des intervalles.

Ecrasement chronologique

Ce comportement correspond à une propagation en suivant l'ordre chronologique des Δ_i. Les figures 7.7 et 7.8 présentent la propagation de la modification d'un Δ_i du membre droit de l'équation 7.1.

Une modification de la valeur Δ (membre gauche) procède de la même manière sur l'ensemble des Δ_i (membre droit). La figure 7.9 présente le résultat d'une modification chronologique pour la structure impliquant deux contraintes d'intervalles de la figure 7.5.

Dans cet exemple, Δ se trouve modifé par l'intervention tardive de Pi. Cette modification va se propager par propagation du membre gauche au travers de l'égalité 7.3. Cette propagation dans le cas chronologique va chercher à modifier les intervalles de du memebre droit de l'égalité dans l'ordre chronologique. Or, au moment où Pi intervient, Δ_3 et Δ_C se déjà écoulé, ont donc pris leur valeur écrite dans la partition et ne sont donc plus modifiables. Δ_{I_2} est en cours d'écoulement et n'est donc plus modifiable. La propagation va donc chercher à modifier Δ_D et Δ_4 dans cet ordre de préférence. Une modification de Δ_D jusqu'à la valeur $\Delta_{D_{max}}$ suffit à rétablir l'égalité entre les deux membres de l'équation imposée par la contrainte d'intervalles. Δ_D prend donc sa valeur maximum et Δ_4 est inchangé.

Ecrasement anti-chronologique

De manière symétrique, l'écrasement anti-chronologique cherche à propager les modifications de valeurs en sens inverse. Les figures 7.3.3.0 et 7.11 présentent la modification en écrasement anti-chronologique de la situation initiale de la figure 7.4.

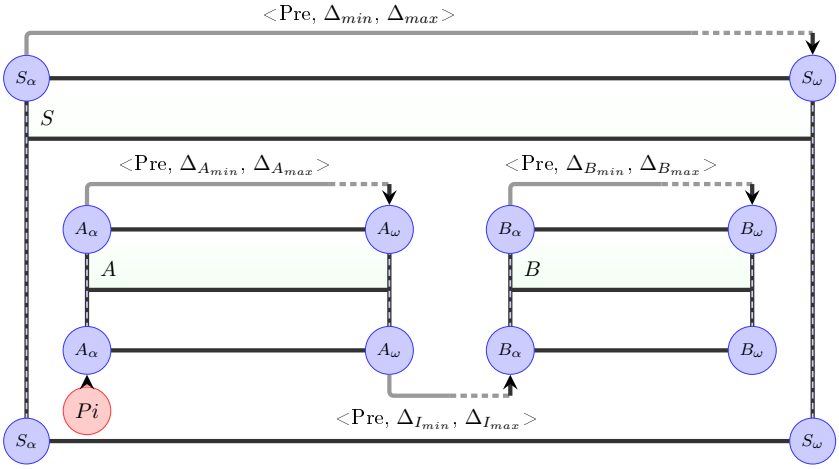
Nous présentons sur la figure 7.12 une modification de la structure de la figure 7.5 selon un écrasement anti-chronologique.

Sur cet exemple, l'intervention du point d'interaction conduit à l'augmentation de Δ, dont la nouvelle valeur va être propagée au travers de la seconde contrainte. La propagation anti-chronologique conduit à modifier Δ_4 et Δ_D. Δ_4 reçoit sa valeur maximum sans que cela soit nécessaire pour rétabli l'égalité imposée par la contrainte. Δ_D est alors augmentée, pour atteindre une valeur Δ'_D, inférieure à $\Delta_{D_{max}}$. Une fois la date de Pi connue, toutes les variables sont fixées.

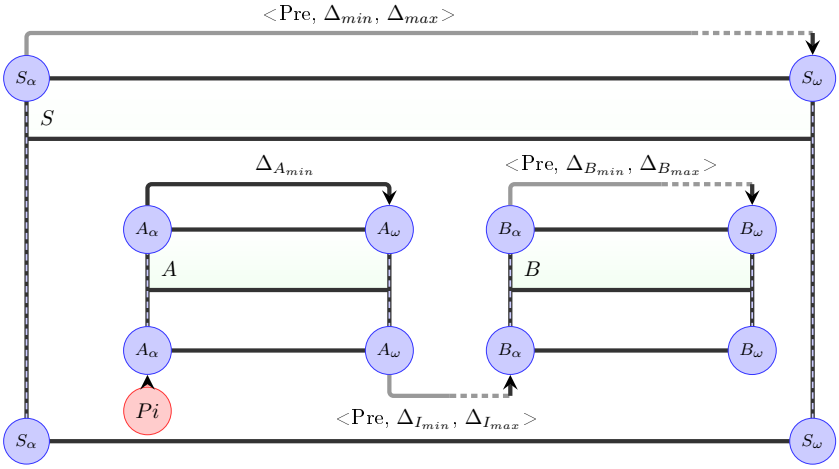
7.3.4 Ecrasement proportionnelle

Les comportements chronologiques et anti-chronlogiques se distinguent par le fait qu'ils cherchent à minimiser le nombre de variables impactées par la propagation d'une modification. Une autre approche consiste à répartir équitablement la modification d'une varibale sur les autres. C'est le propos de du comportement d'*écrasement proportionnelle*. Ce comportement se distingue par une différenciation importante entre les modifications du membre gauche et du membre droit.

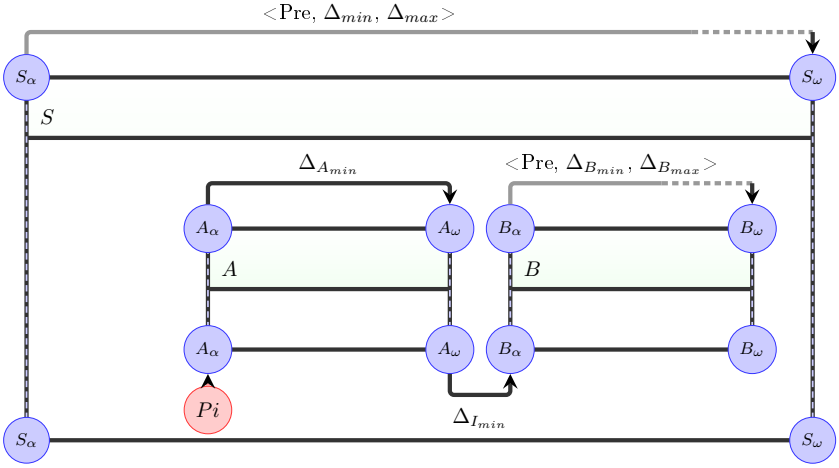
La méthode de propagation pour la modification du membre droit ne modifie jamais le membre gauche et modifie les variables du membre droit pour rétablir l'égalité de la contrainte, proportionnellement aux valeurs de variables écrites (plus la valeur d'une variable est grande, plus elle sera modifiée).



(a) Situation initiale

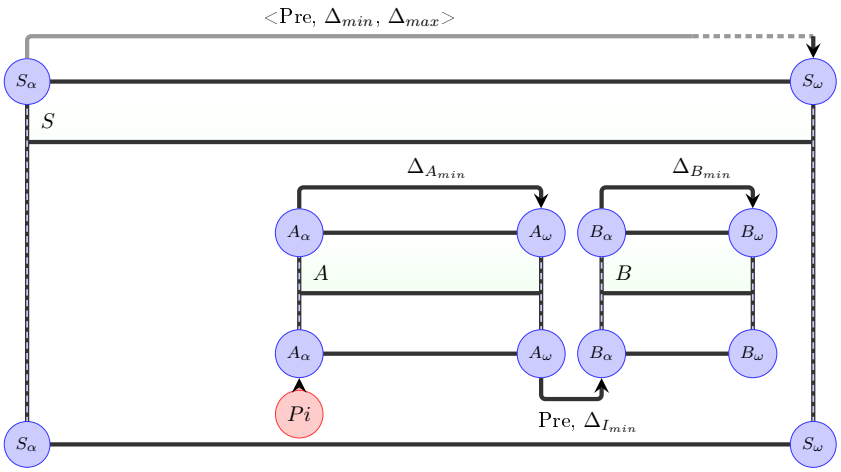


(b) Ecrasement de Δ_A

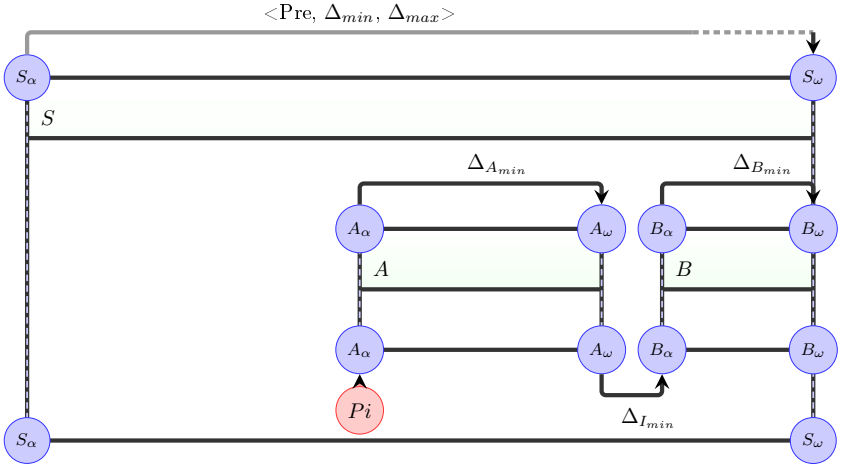


(c) Ecrasement de Δ_I

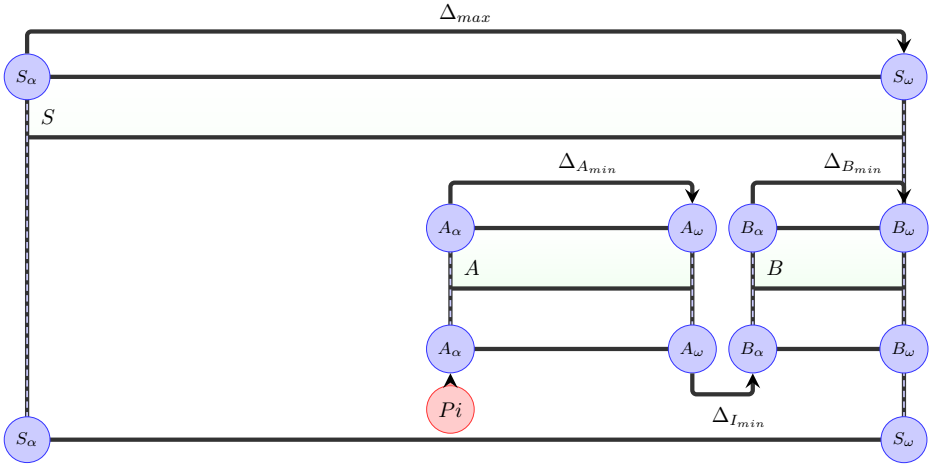
FIG. 7.7 – Exemple de modification avec un comportement *écrasement chronologique*. Comme dans les exemples précédents, les intervalles Δ_1 et Δ_2 ne sont pas représentés. L'augmentation de Δ_1 par le point d'interaction se propage à tous les intervalles de la contrainte selon un ordre chronologique



(a) Ecrasement de Δ_B



(b) Ecrasement de Δ_2



(c) Augmentation de Δ

FIG. 7.8 – Suite de l'exemple d'écrasement chronologique de la figure 7.7. Les derniers intervalles du membre droit de la contrainte sont modifiés. En dernier recours Δ est agrandi.

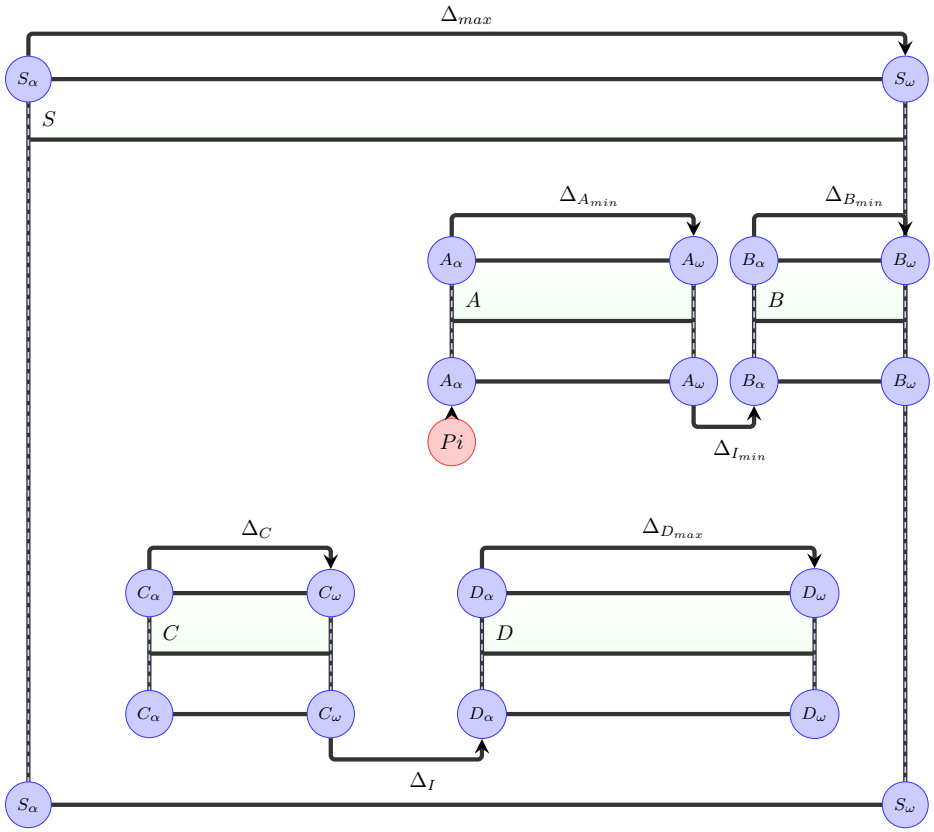
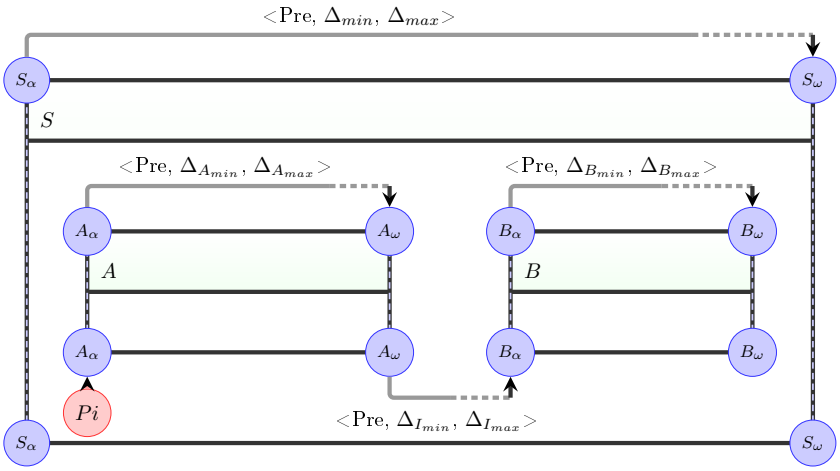
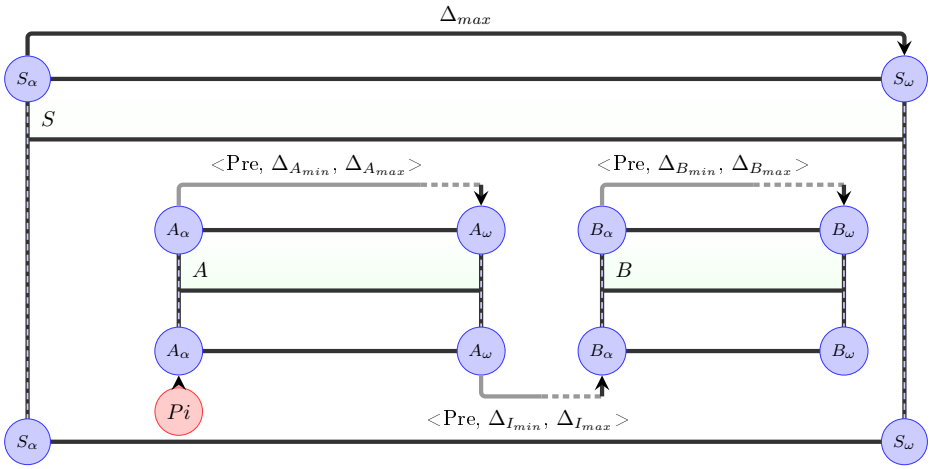


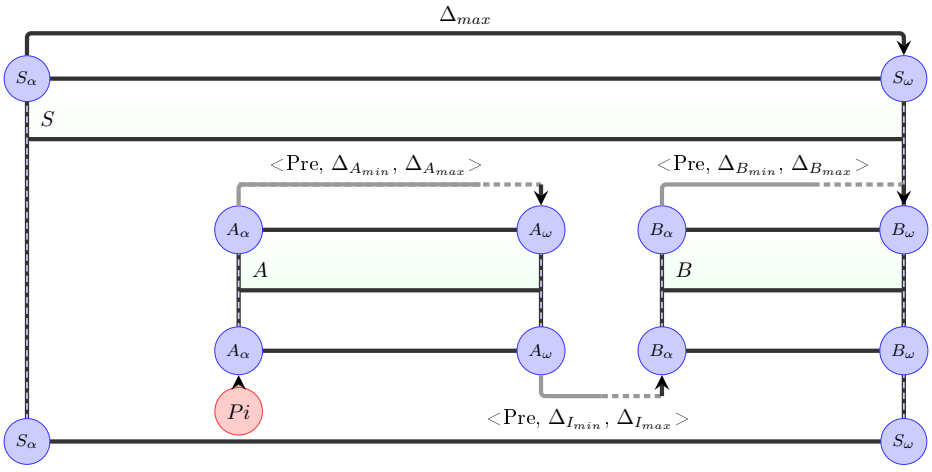
FIG. 7.9 – Une modification avec comportement d'écrasement chronologique de la situation initiale de la figure 7.5. Ici la modification de Δ_1 est telle que Δ est agrandi. La nouvelle valeur de Δ est alors propagée dans la seconde ligne de temps. Etant donnée la date d'apparition de Pi, les intervalles Δ_3, Δ_C et Δ_{I_2} ne sont pas modificables. La propagation chronologique impose de modifier en premier lieu Δ_D qui se trouve alors agrandi de la même manière que Δ, et conduit à adopter la valeur Δ_{max}. La modification de Δ_4 n'étant pas nécessaire, il reste inchangé.



(a) Situation initiale

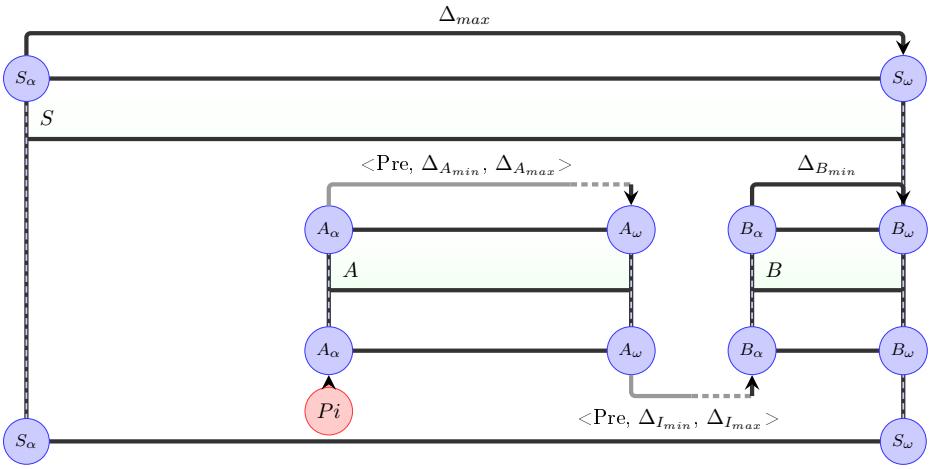


(b) Augmentation de Δ

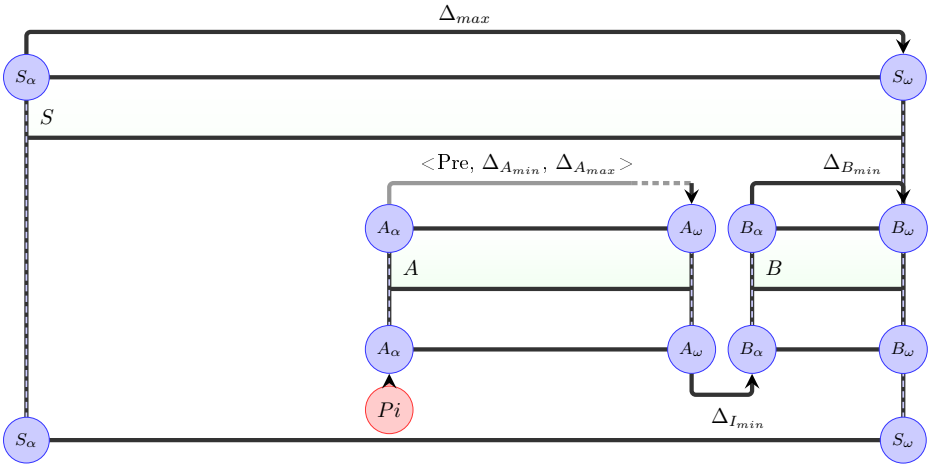


(c) Ecrasement de Δ_2

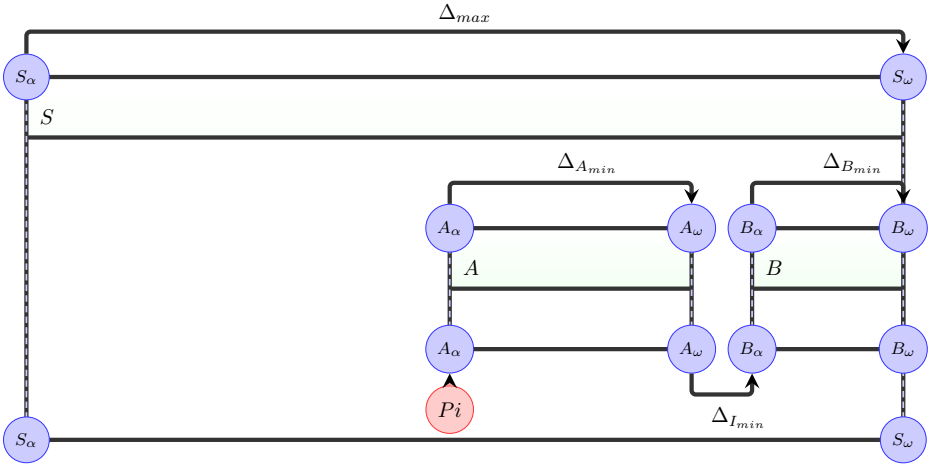
FIG. 7.10 – Exemple de modification selon un écrasement anti-chronologique pour la situation initiale de la figure 7.4.



(a) Ecrasement de Δ_B



(b) Ecrasement de Δ_I



(c) Ecrasement de Δ_A

FIG. 7.11 – Suite de l'exemple de modification selon un écrasement anti-chronologique.

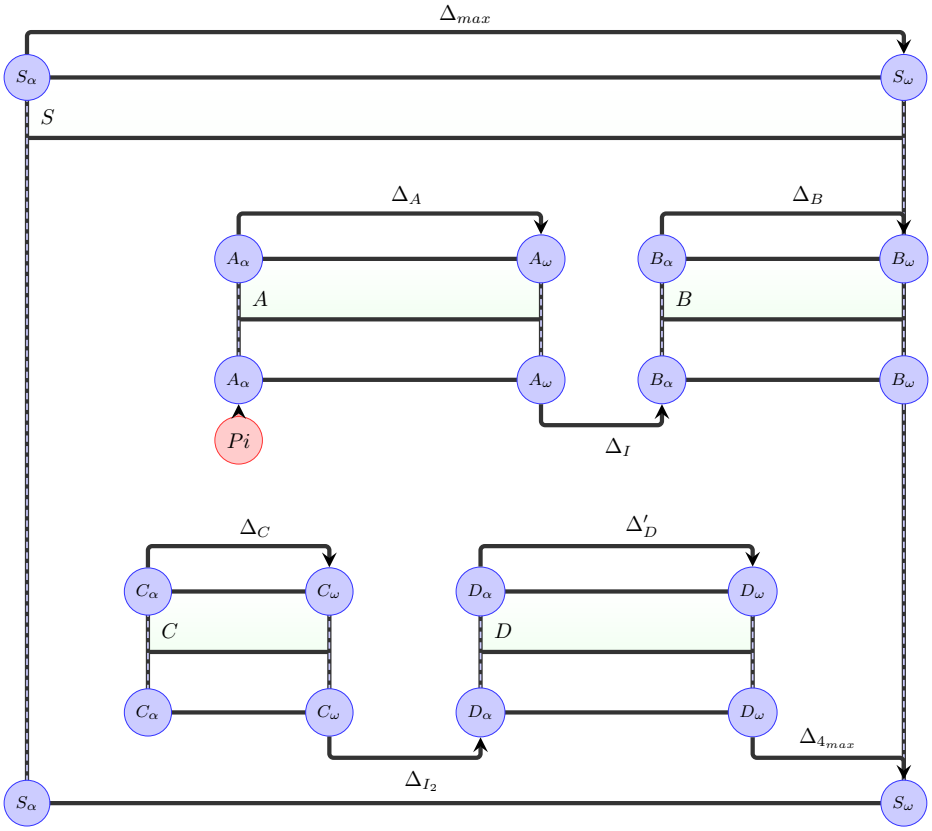
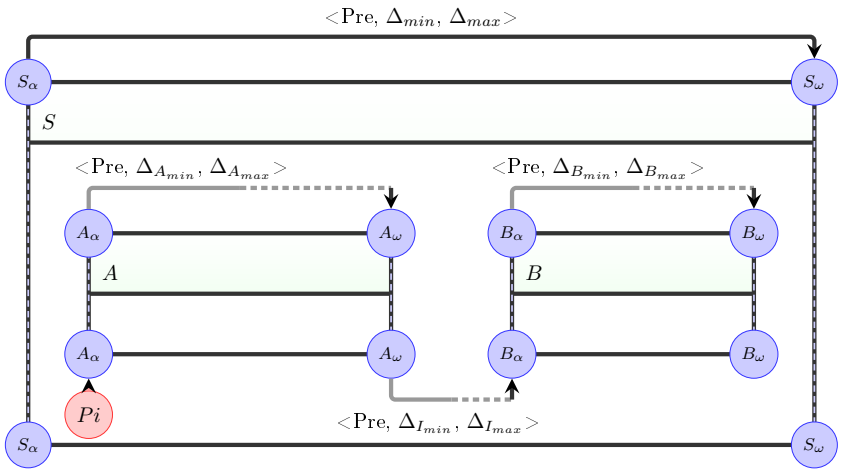
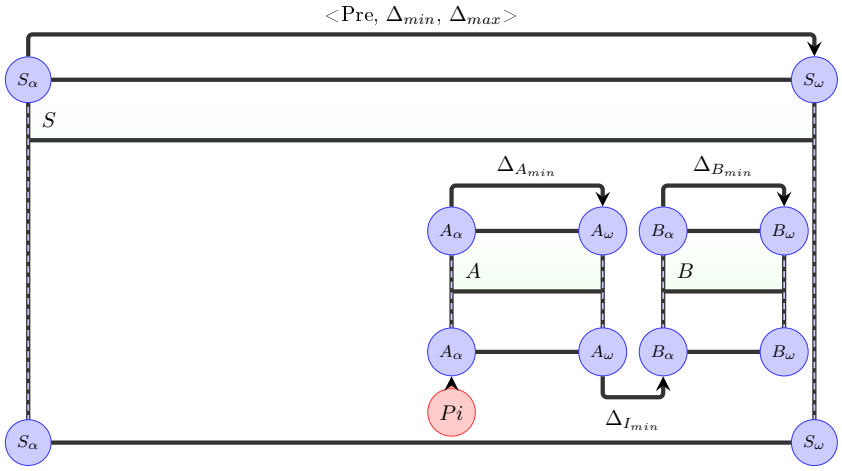


FIG. 7.12 – Une modification avec comportement d'écrasement anti-chronologique de la situation initiale de la figure 7.5. Ici la modification de Δ_1 est telle que Δ est agrandi et Δ_2 totalement réduit. La nouvelle valeur de Δ est alors propagée dans la seconde ligne de temps. Comme il s'agit d'un comportement anti-chronologique, la propagation se fait selon l'ordre de préférence $(\Delta_4 \Delta_D \Delta_{I_2})$, les intervalles Δ_3 et Δ_C n'étant pas modifiables. Δ_4 est alors étendu à son maximum, tandis que Δ_D reçoit une valeur Δ'_D permettant de rétablir l'égalité imposée par la contrainte d'intervalles. Cette valeur étant plus petite que la valeur $\Delta_{D_{max}}$.



(a) Situation initiale



(b) Ecrasement proportionnel

FIG. 7.13 – Un exemple d'écrasement proportionnel. La valeur de Δ n'est pas modifiée même si cela est possible comme ici. L'agrandissement de Δ_1 conduit donc une diminution globale des intervalles modifiables, celle-ci étant répartie proportionnellement sur chacune des variables.

Un exemple est présenté sur la figure 7.13, et la formalisation de cette modification est également présentée dans l'annexe A.

Le cas d'une structure définie avec un comportement proportionnel et disposant de deux lignes de temps, ne présente pas d'intérêt particulier dans la mesure où le membre gauche n'est jamais modifié ; ainsi aucune propagation à d'autres contraintes d'intervalles n'est possible.

La propagation d'une modification du membre gauche se fait de la même manière, proportionnellement aux valeurs des durées des enfants.

Lien avec la modification de tempo

D'aucun pourrait arguer que la propagation du membre gauche de ce comportement est très proche du choix de modification de tempo en réponse à une modification de valeur d'intervalle. Certes, vu des niveaux hiérarchiques supérieurs, une modification du tempo va conduire à des modifications proportionnelles des durées des enfants. Mais du point de vue interne de la structure, un changement de tempo ne va pas induire de modifications de date dans le référentiel de la structure et par conséquent aucune propagation si ces enfants sont eux-mêmes des structures. Seule la vitesse d'exécution sera modifiée.

A l'inverse, dans le cas du comportement proportionnel, les durées seront modifiées et éventuellement et conduiront éventuellement à des modifications de variables plus bas dans la hiérarchie.

7.3.5 Contraintes d'intervalles et synchronisation

Dans les exemples précédents, nous nous sommes restreints à un cas simple où les contraintes d'intervalles d'une structure font intervernir la durée de la structure. Ce n'est pas nécessairement le cas. En effet, il faut se souvenir que les contraintes d'intervalles naissent de la synchronisation de différentes lignes de temps indépendantes en un point donné. Dans les cas présentés jusqu'à présent, le point de synchronisation est la fin de la structure, mais d'autres configurations sont possibles.

La figure 7.14 présente une transformation de l'exemple de la figure 7.5 pour en déplacer le point de synchronisation entre les lignes de temps. Sur cet exemple, les intervalles séparant $start(S)$ et $start(A)$ (Δ_1), $start(S)$ et $start(C)$ (Δ_3), $end(B)$ et $start(S)$ (Δ_2), $end(D)$ et $end(S)$ (Δ_4) ne sont pas représentés pour ne pas surcharger la figure. Ici, la relation de synchronisation entre $start(D)$ et $end(A)$ implique une synchronisation en ce point des lignes de temps (Δ_1 Δ_A) et (Δ_3 Δ_C Δ_{I_2}).

Cette synchronisation induit les contraintes d'intervalles suivantes :

$$\begin{array}{ll} c_1 : & \Delta_{c_1} = \Delta_1 + \Delta_A \\ c_1' : & \Delta_{c_1} = \Delta_3 + \Delta_C + \Delta_{I_2} \end{array}$$

où Δ_{c_1} correspond à l'intervalle séparant $start(S)$ et $end(A)$. Cette variable ne corespond bien sûr pas à la durée d'une structure mais se trouve en membre gauche de ces deux contraintes d'intervalles à cause du point de synchronisation.

De la même manière, on trouvera les contraintes d'intervalles suivantes induites par la seconde partie de la structure S :

$$\begin{array}{ll} c_2 : & \Delta_{c_2} = \Delta_I + \Delta_B + \Delta_2 \\ c_2' : & \Delta_{c_2} = \Delta_D + \Delta_4 \end{array}$$

où Δ_{c_2} correspond à l'intervalle séparant $start(D)$ et $end(S)$.

En effet le point de synchronisation en $end(A)$ induit la naissance de deux lignes de temps partageant cette origine : ($\Delta_I \Delta_B \Delta_2$) et ($\Delta_D \Delta_4$). Ces deux lignes se synchronisant en $end(S)$.

Enfin, le lien avec la variable Δ se fait au travers de la contrainte d'intervalle suivante :

$$c_3 : \Delta = \Delta_{c_1} + \Delta_{c_2}$$

Rappelons au passage que la définition d'un comportement d'intervalles étant globale pour toute une structure, les cinq contraintes définies ci-dessus ont toutes le même comportement : celui défini pour S. Signalons également que les variables Δ_{c_1} et Δ_{c_2} ne correspondant pas à la durée d'une structure, il ne peut être question de modifier un quelconque tempo suite à une modification de leur valeur.

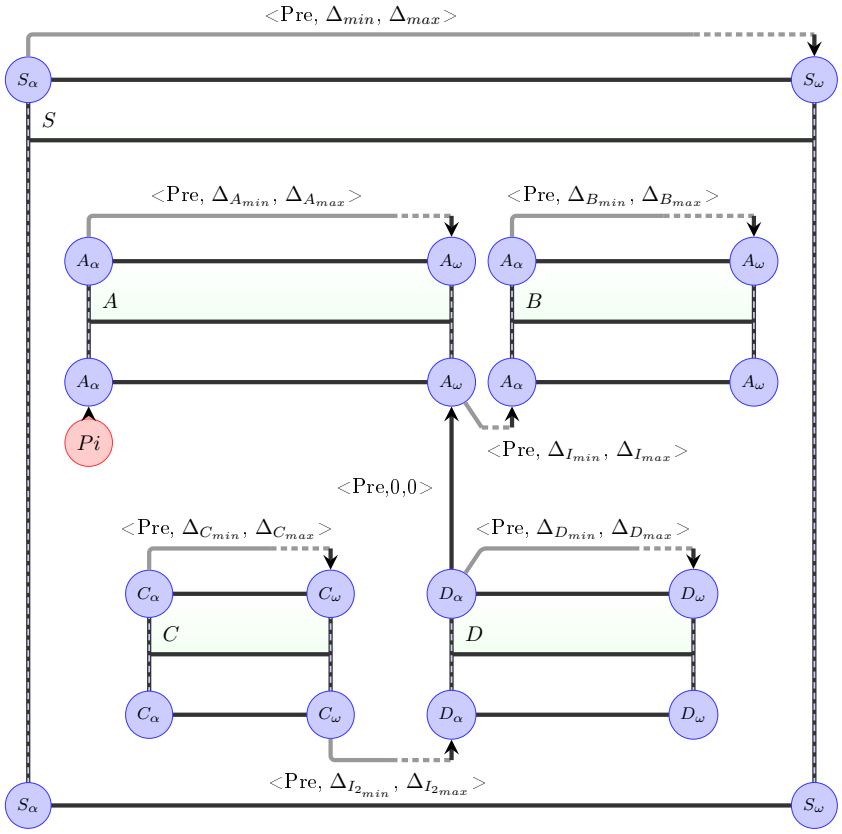


FIG. 7.14 – Une situation dans laquelle l'ordre partiel entre les événements d'une structure implique la définition de deux contraintes d'intervalle pour cette structure. Par concision, certains intervalles ne sont pre-présentés : Δ_1 sépare $start(S)$ et $start(A)$, Δ_2 sépare $end(B)$ et $end(S)$, Δ_3 sépare $start(S)$ et $start(C)$, et enfin Δ_4 sépare $end(D)$ et $end(S)$. On peut considérer que les listes d'intervalles $(\Delta_1\Delta_A\Delta_I\Delta_B\Delta_2)$ et $(\Delta_3\Delta_C\Delta_{I_2}\Delta_B\Delta_4)$ forment deux lignes de temps inédependantes devant se synchroniser en $end(S)$.

Remarque 7.2 *La séparation en lignes de temps que nous la proposons ici n'est pas l'unique possibilité de décomposition. On peut s'intéroger sur ce choix. Nous décomposerons systématiquement les suites d'intervalles selon les lignes de temps les plus courtes, c'est à dire entre deux points de synchronisation directement successifs. Cette option nous est imposée par la définition des comportements que nous nous sommes donnés. En effet, on peut voir sur les exemple de modifications impliquant plusieurs contraintes d'intervalles (7.9, 7.12), que ceci nécessite de disposer des contraintes d'intervalles correspondant aux sommes d'intervalles contenant un minimum de variables. Cette remarque doit rester à l'esprit du lecteur, car elle sera centrale lors de l'exposition des modèles opérationels d'exécution des partitions (chapitre 9).*

Domaines de variables

Dans tous les exemples de comportements et de modifications de variables temporelles que nous venons de présenter, nous avons fait un usage important des valeurs minimum et maximum associées aux variables. Sans trop approfondir cette question pour l'instant, ils nous faut signaler que ces valeurs extrêmes des variables conditionnent fortement l'existence d'une solution pour le système de contraintes temporelles. Derrière la manipulation désinvolte que nous en faisons, se cache la cohérence du système de contraintes impliquant la réduction des domaines des variables. Cette notion est très importante au regard de caractéristiques :

- les relations temporelles (et donc leurs valeurs extrêmes associées) sont définies par le compositeur, or lui demander d'assumer la cohérence des relations qu'il introduit ne parait réaliste.
- les variables que nous faisons émerger dans les contraintes d'intervalles non liées à la durée d'une structure (c_1 et c_2 dans l'exemple ci-dessus), doivent disposer de domaines cohérents avec l'existence d'une solution du système de contraintes.

Ces questions seront évoquées plus avant dans la suite, dans la section 7.5.1 concernant la jouabilité des partitions, et dans le chapitre 8 portant sur l'outil d'édition des partitions.

7.4 Point d'interactions et structures logiques

Pour l'instant, nous nous sommes intéressés uniquement à l'ajout de points d'interaction dans les structures linéaires. Il est bien sûr possible de définir des points d'interaction dans les structures logiques. Les considérations sur les intervalles de temps sont dans ce cas caducs, et ces points d'interaction ont pour rôle d'orienter les choix au niveau des relations logiques. Précisons briévement cette utilisation des points d'interaction dans les structures logiques au travers d'un exemple, celui de la figure 7.15.

Rappelons que les structures et relations logiques sont définies dans les sections 5.5.1 et 5.6. Nous reprenons d'ailleurs ici, l'exemple de la structure logique utilisé lors de leur définition, auquel nous avons ajouté un point d'interaction.

En notant rl_1, la relation logique liant $end(S_2)$ à $start_(S_2)$ et $start(T_1)$, rappelons que cette relation dispose d'un intervalle de temps Δ_1 associé à des valeurs extrêmes $\Delta_{1_{min}}$ et $\Delta_{1_{max}}$. De plus, a et b représentent des expressions logiques impliquant des variables de S_1. L'ajout du point d'interaction Pi sur $start(T_1)$ doit s'interpréter de la manière suivante : à la fin de S_2, après l'écoulement de la durée $\Delta_{1_{min}}$ si b est vrai alors Pi est activable pour déclencher le début de T_1.

Quelques remarques à présents, dans notre exemple comme le $start(S_2)$ n'est pas dynamique, alors pour peu que a soit vrai après l'écoulement de $\Delta_{1_{min}}$, S_2 sera immédiatement déclenché car le temps d'activation de Pi sera alors réduit à 0. Pour palier à cette situation inconfortable, on peut rendre $start_{S_2}$ dynamique grace à un nouveau point d'interaction, et proposer ainsi d'effectuer le choix de l'objets déclencher au travers des deux points d'interaction. On peut également utiliser le choix par défaut de la relation rl_1 , en posant $a = faux$ et en définissant le bouclage sur S_2 comme choix par défaut de rl_1, alors après l'écoulement de $\Delta_{1_{min}}$, si b est vrai, Pi se déclenchable. Le bouclage automatique vers $start(S_2)$ sera impossible puisque a sera faux. En revanche, si Pi n'a pas été déclenché lorque la durée maximum $\Delta_{1_{max}}$ est écoulée, alors c'est le bouclage qui sera choisie. En effet, dans le cas limite, le choix pas défaut est pris quelle que soit la valeur de l'expression logique qui lui est associée. On peut ainsi définir des réactions "d'urgence" en cas de non délenchement d'un point d'interaction.

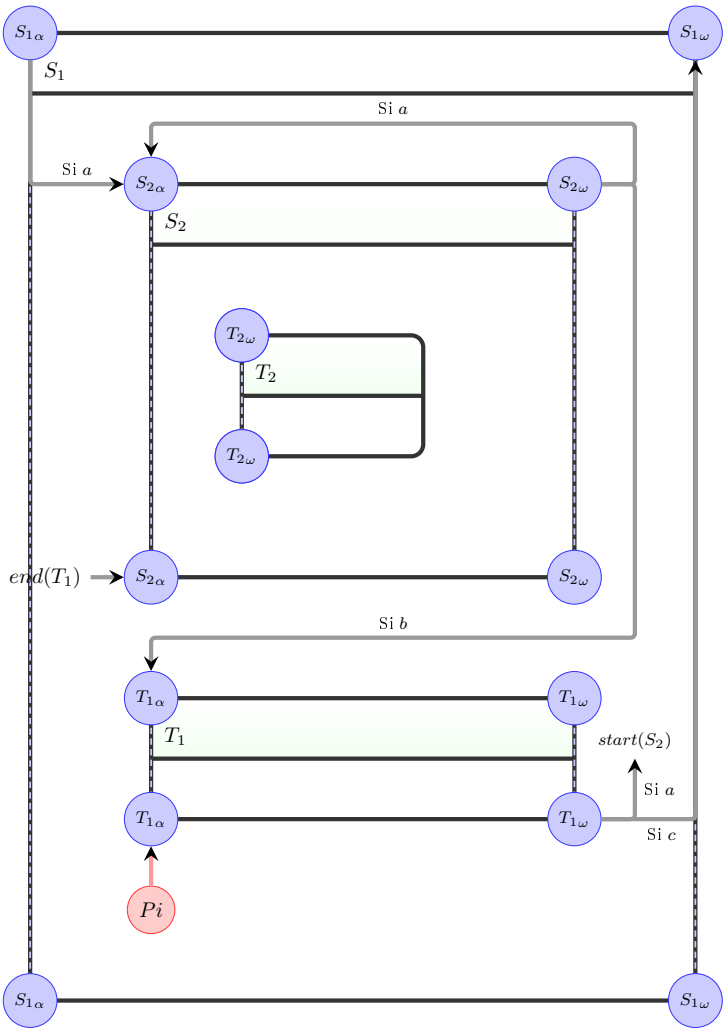


FIG. 7.15 – Un exemple de structure logique dans laquelle un point d'interaction a été ajouté. Ici, $start(T_1)$ est rendu dynamique par Pi. La relation logique rl_1 sortant de $end(S_2)$ dispose d'un intervalle Δ_1 avec des valeurs extrêmes $\Delta_{1_{min}}$ et $\Delta_{1_{max}}$. Quand $\Delta_{1_{min}}$ s'est écoulé après $end(S_2)$, les transitions associées à rl_1 sont valides. A ce moment, si b est vrai, Pi devient actif et peut être déclenché. Cependant, si a est vrai le bouclage sur S_2 sera déclenché car la période d'activation de Pi sera réduite à zéro.

Le choix impliquant le point d'interaction peut également être défini comme choix par défaut. Dans ces conditions, le système déclenchera automatiquement le point après écoulement de $\Delta_{1_{max}}$.

7.5 Exécution

Il nous faut maintenant formaliser ce qu'est une exécution d'une partition interactive. Intuitivement, une exécution est une instanciation des dates absolues de tous les événements de la partition vérifiant les contraintes temporelles.

Définition 7.4 *Soient P une partition et exe une exécution de P, nous définissons comme une relation sur l'ensemble des événements de P, les entiers et les réels :*

$$exe \subseteq \Sigma_P \times \mathbb{N} \times \mathbb{R}$$

t.q. :

$$(e, i, d) \in exe \Leftrightarrow date\_occ(e, parent(e), i) = d$$

De plus comme les valeurs de dates respectent les relations temporelles :

$$\begin{array}{l} rt(pre, e_1, e_2, \Delta_{min}, \Delta_{max}) \Rightarrow \\ \forall i \in \mathbb{N}, (e_1, i, date_{1_i}), (e_2, i, date_{2_i}) \in exe \Rightarrow \\ \qquad \Delta_{min} \leq date_{2_i} - date_{1_i} \leq \Delta_{mix} \end{array}$$

La même propriété étant valable pour les relations post.

De plus, nous distinguerons exécutions partielles et exécutions totales de la façon suivante : Une exécution exe d'une partition P sera dite *totale* si sa projection sur la première coordonnée (les événements) est égale à l'ensemble de tous les événements de la partition :

$$\Pi_1(exe) = \Sigma_P$$

et *partielle* si :

$$\Pi_1(exe) \subseteq \Sigma_P$$

7.5.1 Notion de *jouabilité* et validité des points d'interaction

Cette définition d'une exécution nous conduit à nous pencher sur les choix pris par le musicien pendant l'interprétation d'une pièce. Nul besoin d'être grand clerc pour s'apercevoir qu'il est possible de choisir des dates de déclenchement ne respectant pas les contraintes de la partition.

L'exemple le plus direct est certainement celui de deux événements interactifs dont le musicien intervertirait l'ordre imposé par une relation temporelle. Mais il est possible d'invalider des contraintes temporelles bien plus subtilement. En chosissant des dates ne respectant pas les valeurs Δ_{min} et Δ_{max} d'une relation temporelle, ou encore en choisissant la date d'un événement interactif de telle manière que plus tard au cours de l'exécution, une contrainte ne pourra plus être respectée.

Pour caractériser l'existence d'une exécution, nous définissons la notion de *jouabilité* d'une partition.

Une partition P sera dite *jouable*, si il existe une exécution de P. Cette définition est étendue aux partitions en cours de déroulement, dans ces conditions, la partition P sera considérée *jouable* si il existe une exécution de P acceptant les dates d'événements dynamiques déjà choisies.

On peut formaliser cette notion à l'aide des éxécutions partielles et totales :

Définition 7.5 *Soit P une partition intéractive et exe_{part} une exécution partielle de P, alors P sera dite* jouable *étant donnée exe_{part} si il existe une exécution totale exe_{tot} de P telle que :*

$$exe_{part} \subseteq exe_{tot}$$

En particulier, la partition écrite (avant début du jeu) correspond à une exécution partielle exe telle que :

$$\Pi_1(exe) = \emptyset$$

Plusieurs attitudes sont envisageables compte tenu de cette définition. Lors du déroulement d'une partition, on peut faire en sorte que la jouabilité de la partition soit toujours assurée. Ce choix implique de limiter les actions du musicien et de n'accepter que celles qui débouchent sur une exécution totale de la partition ; deux cas peuvent alors se produirent : une tentative de déclencher un point d'interaction trop tôt induiera, la non réaction du système à l'action du musicien, tandis qu'un retard trop important conduira au déclenchement automatique du point d'interaction. Dans la suite, nous opterons pour cette stratégie.

Cependant, les conséquences de cette approche peuvent être déroutantes pour le musicien, voire désservir l'objectif du compositeur. En effet, certaines pièces peuvent être sciemment écrites pour comporter l'éventualité d'une impasse, l'impossibilité de mener une exécution à son terme. Nous proposerons donc également d'autres stratégies au moment de présenter le système d'éxécution des partitions.

7.5.2 Interruption de structures

L'organisation hiérarchique des structures implique des situations dans lesquelles un point d'interaction peut être disposé sur la fin d'une structure, autorisant le musicien à interrompre l'exécution de la structure.

La figure 7.16 présente une telle situation.

Dans ce cas, il est possible pour le compositeur de choisir si il souhaite que le contenu de la structure aille à son terme, ou si il en autorise l'interruption avant que tous ses objets enfants aient été exécutés.

On peut noter que la possibilité d'interrompre une structure peut conduire à des situtations dans lesquelles un événements impliqué dans une relation temporelle ne se produit pas. Dans l'exemple de la figure 7.16, si le compositeur autorise l'interrruption de la structure S_2 avant l'exécution de tous ses objets enfant, et que le point Pi est déclenché avant l'occurence de $end(E_2)$, la relation temporelle liant cet événement à $end(E_4)$ n'a plus lieu d'être.

Dans le cas des structures logiques, intedire une interuption de la structure avant que tous ses enfants n'aient été exécutés n'aurai pas grand sens. Le choix se fait alors entre la possibilité d'interrompre la structure pendant son exécution, et devoir attendre de se trouver dans l'intervalle de validité d'une relation logique conduisant à quitter la structure. Formellement, en reprenant la comparaison avec les automates, cela signifie que la séquence de choix effectués depuis le début de la structure doit former un mot reconnaissable par l'automate associé à la structure logique.

Dans le cas de structures présentant une boucle entre leurs objets enfants, cette caractéristique peut conduire à un comportement un peu particulier.

En effet, si un objet enfant dont la fin est reliée à celle de la structure par une relation logique fait partie d'une boucle, alors en posant un point d'interaction sur la fin de cette structure et en faisant le choix d'imposer le déroulement du contenu avant terminaison, on rend le point d'interaction final valide à chaque fin de déroulement de boucle mais pas pendant. En l'absence de déclenchement, le point d'interaction sera donc valide par intermittence.

La figure 7.17 décrit pareil exemple. Sur cet exemple, la structure S_1 est logique. Elle contient une structure temporelle S_2 et les relations logiques définies permettent un bouclage sur S_2. De plus, la relation logique définie entre $end(S_2)$, $start(S_2)$ et $end(S_1)$ est telle que le choix par défaut conduit à répéter S_2. Rappelons que les relations logiques (definies dans la section 5.6) dispose d'un intervalle de temps qui force un choix par défaut lorque sa valeur maximum est atteinte sans qu'un choix n'aie été effectué. Ainsi dans notre exemple, à chaque fin de S_2, la possibilité de choisir entre $end(S_1$ et $start(S_2)$ est offerte pendant un laps de temps déterminé, lorsque ce dernier est écoulé, la structure S_2 est automatiquement exécutée de nouveau.

En outre, la structure S_1 est incluse dans une structure linéaire S_0 dans laquelle un point d'interaction déclenchant $end(S_1)$ est défini. Le compositeur a de plus choisi de ne pas autoriser l'interruption de S_1. Par conséquent, seul le déclenchement de Pi au sortir de S_2 est possible. La validité de Pi est donc limité à l'intervalle de la relation logique entre $end(S_2)$ et $end(S_1)$. Comme cette situation est amenée à se répéter à cause de la boucle sur S_2, on a alors une validité périodique de Pi.

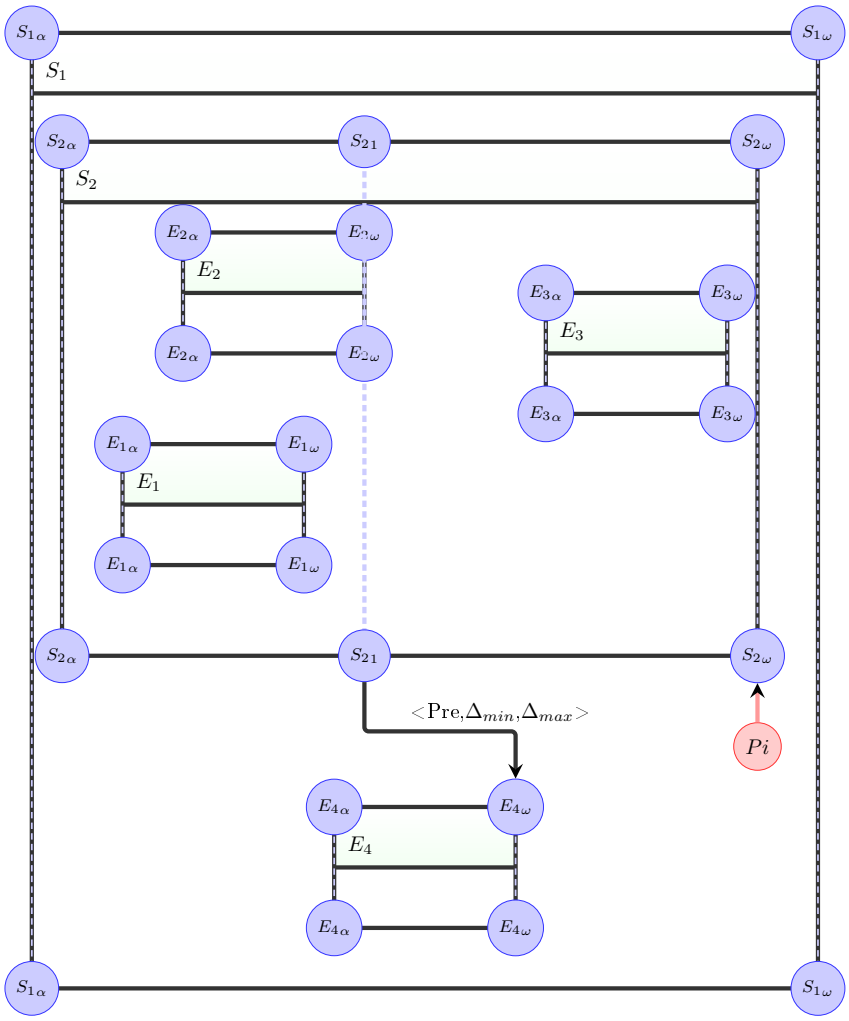


FIG. 7.16 – Un exemple de structure linéaire dont un point d'interaction peut interrompre le déroulement. Dans le cas présent, un déclenchement de Pi avant $end(E_2)$ peut conduire à invalider la relation temporelle.

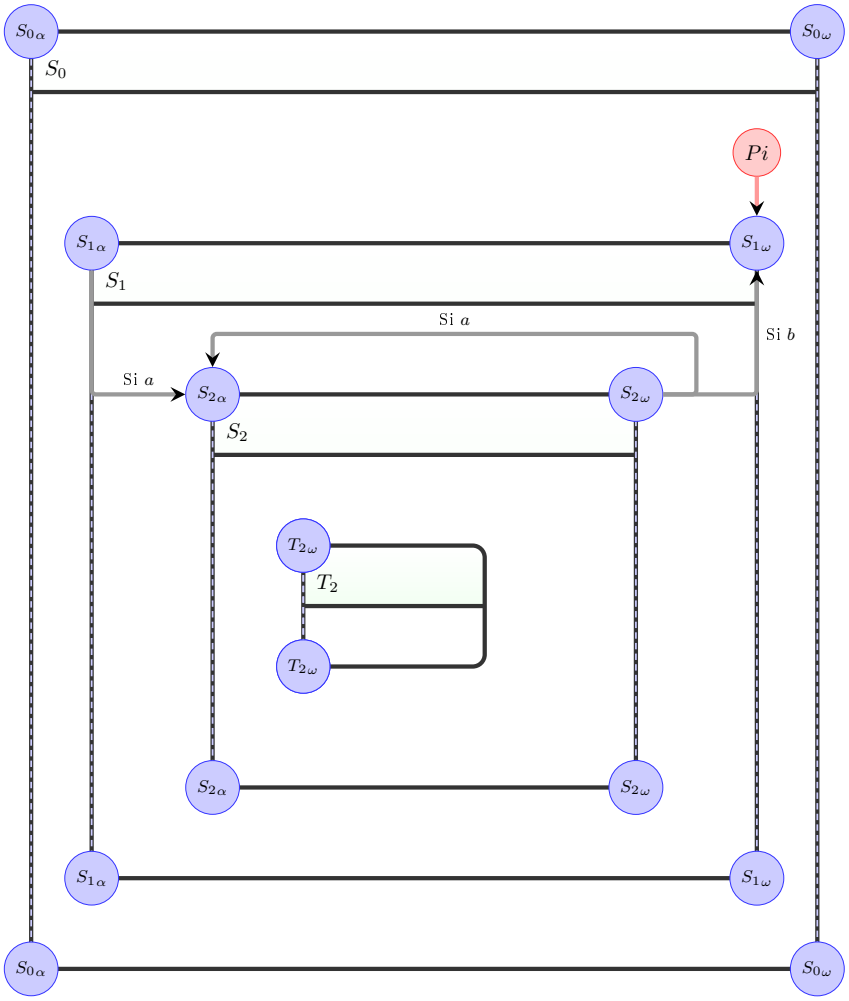


FIG. 7.17 – Un exemple de structure logique avec une fin interactive, conduisant à une validité alternative du point d'interaction. En effet dans cet exemple, nous suppposons que le compositeur a spécifié que la structure logique S_1 ne peut être interrompue en cours d'exécution de l'un de ses objets enfant. Pi ne peut donc être actif que lorsque S_2 est terminé, pendant l'intervalle de valité associé à la relation logique rl sortant de S_2. Or nous supposons que le choix par défaut associé à cette relation est le bouclage sur S_2. Par conséquent si Pi n'est pas déclenché avant l'écoulement de Δ_{max}, durée maximum associé à rl, alors le système bouclera automatiquement sur S_2 rendant Pi inactif jusqu'à la fin de la nouvelle exécution de $end(S_2)$, cettte situation conduit donc à une validité de Pi par intermitence.

Troisième partie

Modèle opérationnel pour l'exécution des partitions

Résumé

Dans cette partie, nous proposons une modélisation informatique des partitions interactives écrites dans le formalisme de la partie précédente. Cette modélisation vise à fournir une description informatique de l'exécution de ces partitions. Parmi les différents outils mis à notre disposition par l'informatique théorique, nous avons opté pour les réseaux de Petri et plus précisément les réseaux de Petri à flux temporels. Les capacités de ces derniers à modéliser des systèmes faiblement synchrones impliquant des contraintes sur les intervalles de temps séparant leurs événements, s'accordent assez bien avec les caractéristiques des partitions. Notons que nous avons envisagé d'utiliser d'autres formalismes comme $NTCC$[18]. Sans développer l'utilisation de ce formalisme, nous adoptons certains de ses éléments dans les solutions que nous présentons ici.

Pour modéliser l'exécution des partitions, nous abordons notre formalisme comme un langage de programmation, les partitions étant alors vues comme des programmes écrits dans ce langage. Cette approche nous permet dans le premier chapitre de définir formellement une machine abstraite, la machine ECO, capable d'exécuter ces programmes.

Nous proposons ensuite un modèle d'exécution des partitions représentées sous forme de réseau, au sein d'une machine ECO adaptée à cette représentation.

Abstract

In this part, we propose a computer model for the interactive scores written in the formalism presented in the previous part. We aim to provide a computer description of the execution of these scores. We choose to use the Petri nets among other tools, and more precisely, the Time Stream Petri nets. They allow to model low-synchronized system with specific temporal constraints on the time-intervalls between their events. Therefore, there are well indicated to model the interactive scores. Note that we investigated the use of other formalisms like $NTCC$[1]. Even if we do not fully develop this solution, some of its elements can be recognized in the solutions that we present here.

In order to modalize the execution of the scores, we approach our formalism as a programmation language. From this point of view, a score is a program written in this language. This approach allows us in the first chapter to formally define an abstract machine, called the ECO machine, able to execute these programs.

Then we propose a model for the execution of the scores and a description of an ECO machine adapted to this representation.

[18] Non deterministic Temporal Concurrent Constraint calculus

Chapitre 8

Outils d'édition et machine d'exécution

La présentation du formalisme des partitions interactives, bien qu'orientée par l'approche musicale de ce dernier, a largement insisté sur certaines notions algorithmiques qui sous-tendent la construction d'une partition. Comme nous l'affirmions dans les chapitres précédents, notre formalisme permet d'écrire dans une large mesure les calculs qu'implique la réalisation d'une partition. Naturellement, pour une écriture complète des calculs, il nous faudrait élargir le formalisme pour permettre la description des processus.

Cependant, il est légitime de voir dans notre modèle les prémisses d'un langage de programmation pour l'écriture de partitions interactives, vues comme des algorithmes dont l'exécution serait temporisée et contrôlée par le musicien au cours de l'interprétation.

Au regard de cette approche, les fonctions d'un outil de composition assistée pour l'écriture des partitions interactives, seraient assez proches de celles dont disposent les compilateurs comme les analyses lexicale et syntaxique. De la même manière, un système interactif permettant à un interprète de jouer une partition s'apparenterait à une machine exécutant un code compilé depuis cette partition "source".

C'est l'optique que nous adoptons dans ce chapitre pour évoquer d'une part les fonctionnalités dont pourrait disposer une application d'édition de partitions interactives, ainsi que pour définir le système permettant d'exécuter les partitions.

8.1 Un langage de programmation

Si notre formalisme constitue un langage de programmation, on peut s'intéresser aux paradigmes qu'il permet de manipuler ou plus modestement aux caractéristiques qu'il partage avec des langages existants.

Il pourrait sembler curieux d'avoir construit un langage de programmation avant d'en étudier les caractéristiques et possibilités. Cependant, notre formalisme est avant tout le fruit d'une certaine modélisation de la musique et de l'interaction avec elle. Certes son élaboration fut orientée, et il est assez rassurant de constater que nous avons bien abouti à un langage visuel s'appuyant sur la programmation par contraintes. Cependant certains aspects du modèle ont émergé au cours de sa création.

On peut ainsi citer :

- **la programmation concurrente** : fonder notre modèle sur l'agencement de processus asynchrones nous conduit immanquanblement vers la prise en compte de la concurrence. L'écriture des branchements et les éventuelles situations de famine ou de bloquage qu'elle implique ancrent profondément notre formalisme dans ce paradigme. Bien que l'absence de description formelle des processus limite quelque peu une vérification approfondie au moment de l'édition, certains outils de la programmation concurrente pourraient être mobilisés pour détecter d'éventuels incompatibilité dans le déroulement concurrent des processus.
- **la programmation événementielle** : l'écriture de l'interaction au travers des points d'interaction implique que l'exécution d'une partition sera grandement conditionnée par l'attente de messages extérieurs au système. Lors de l'édition, il peut donc être judicieux de veiller à la validité de ces messages et aux éventuelles interférences entre eux. Tandis qu'à l'exécution, la cadence d'évolution du système devra s'harmoniser avec la fréquence d'apparition de ces événements.

– **la description formelle de systèmes temporels** : bien qu'il ne s'agisse pas à proprement parler d'un paradigme de programmation, nous ne pouvons passer sous silence les points communs entre notre modèle, et les formalismes de système temporisé. Le plus connu est certainement RT-Lotos[19] [30] que ses concepteurs décrivent comme "un langage permettant de décrire formellement les systèmes contraints par le temps". Les outils associés à ce langage incluent la vérification et la simulation d'un système spécifié, mais également la compilation d'une spécification en un automate temporisé de type DTA[20].

Dans notre définition d'outils d'aide à la composition et de la machine d'exécution, nous garderons à l'esprit la similarité avec ces formalismes pour nous inspirer des outils et mécanismes élaborés dans leur cadre.

8.2 Edition

Nous l'avons évoqué, l'aide à la composition d'une partition interactive vue comme un programme se rapproche de l'utilisation d'analyseurs lexicaux et synthaxiques. Cette vérification des propriétés (temporelles, d'interaction...) définies par le compositeur doit s'accompagner d'une aide dans la démarche de création de la situation initiale de la partition. En effet, nous avons considéré la partition comme un système contraint placé dans une situation d'équilibre qui va être perturbée pendant l'exécution. Il convient d'épauler le compositeur dans l'élaboration de cette situation dans laquelle les contraintes sont satisfaites. Nous proposons ici de nous arrêter sur les points clés de l'aide à la composition de partitions interactives, sans décrire formellement les mécanismes qu'ils mettent en œuvre, ni les considérer comme les fonctionnalités d'un logiciel d'édition.

8.2.1 Les relations temporelles

Les relations temporelles définies entre les points de contrôle sont en première ligne de l'aide à la composition et la vérification. Le système doit veiller à ce que les relations soient maintenues lors de l'édition, ou éventuellement indiquer au compositeur que l'une d'elles n'est pas vérifiée.

Les situations d'invalidation de relations sont les suivantes : le compositeur modifie la date d'un point de contrôle entraînant ainsi l'invalidation d'une relation impliquant ce point de contrôle, le compositeur cherche à imposer une relation temporelle entre des événements dont les dates ne vérifient pas cette relation.

Dans ces conditions, le système peut soit prévenir le compositeur de l'incompatibilité de sa volonté avec les relations ou valeurs de date courantes, soit automatiquement instancier de nouvelles valeurs de date pour des événements, respectant la date modifiée ou la nouvelle relation et les relations déjà présentes dans la partition. La première stratégie n'est intéressante que si le système identifie la cause du blocage afin de proposer au compositeur d'y remédier. Tandis que la seconde est susceptible d'instancier une nouvelle solution modifiant nombre de dates courantes, sans que le compositeur puisse réellement percevoir à l'avance les conséquences d'une modification. Des règles de priorité entre les différentes dates d'événements, proches des méthodes de propagation présentées précédemment, pourraient éventuellement orienter le calcul des nouvelles valeurs le rendant ainsi plus prévisible, voire conforme à une volonté du compositeur.

Quelle que soit la stratégie retenue, il est nécessaire de disposer pour la composition d'un système de résolution du problème de contraintes formé par les dates d'événement et les relations, capable de calculer des solutions et d'en expliquer l'absence. Notons au passage, que la représentation de ce problème de contraintes à des fins de résolution n'est pas nécessairement basé sur les relations temporelles. Si un formalisme de description du problème s'avère plus efficace pour la résolution, il est préférable de l'utiliser et de réserver les relations temporelles pour la composition.

Cette remarque est plus pertinente encore à propos de la hiérarchie et des relations implicites. Il est probable qu'une description des contraintes temporelles particulière rend la prise en compte des relations

[19] Real Time Lotos
[20] Dynamic Timed Automaton

implicites plus aisée à l'édition. Cependant, la mise en lumière de ces relations dans notre formalisme à destination du compositeur reste nécessaire. Pour ce faire, il conviendrait d'utiliser un outils de description de problèmes d'ordonnacement comme par exemple les graphes AOA [48] ou graphes d'activité. Il s'agit de graphes orientés dans lesquels, les nœuds représentent les événements et le poids des arcs portent les intervalles de temps séparant les événements.

Sur le graphe, on peut facilement déduire les contraintes temporelles entre deux événements. Il suffit de calculer un chemin entre les deux nœuds. Un chemin empruntant des arcs tous dans le même sens implique l'existence d'une relation temporelle. Dans le cas contraire la recherche du plus proche ancètre dans le graphe [8] permet d'exprimer les contraintes temporelles en terme de relations.

A titre d'exemple, nous reprenons la partition utilisée dans le chapitre 6 pour illustrer la synthèse de relations temporelles implicites sur la figure 8.1. La graphe AOA associé est également présenté. Sur cet exemple, le compositeur a décidé de faire apparaître les points de contrôle $end(O_1)$ et $end(0_2)$ au niveau de S. Etant données les relations temporelles présentes dans S, ces deux points sont contraints par les valeurs que peut prendre l'intervalle de temps les séparant. Mais exprimer cette contrainte à l'aide de relations temporelles passent par la détection automatique d'un point de contrôle. Pour identifier un tel point, on peut utiliser un graphe AOA et y détecter un ancre commun au deux points. Plusieurs points peuvent convenir, on peut alors choisir le plus proche ($start(O_1)$.

Notons que l'existence d'un chemin entre deux événements et d'un ancètre commun est assurée par le début de la structure parent et les relations de cohérences. En outre, cette utilisation implique la mise à jour et le maintien des valeurs des pondérations des arcs, en fonction des modifications du compositeur.

Ce type de structure semble également convenir à la mise en avant de points de contrôle stratégiques.

8.2.2 Résolution des contraintes globales

De la même manière qu'il propose des solutions pour les variables de dates, l'outil d'édition doit être capable d'instancier des valeurs pour les autres variables des structures. Etant donnés les liens entre celles-ci et les dates des événements, la manipulation d'un seul système de contraintes impliquant toutes les variables semble la solution la plus réaliste.

Cependant, les contraintes impliquant les entrées et sorties d'objet temporels nécessitent une description formelle des processus pour être intégrées dans un tel système de résolution.

On peut noter que la représentation de la hiérarchie dans le système de contraintes (CSP hiérarchique) pourrait faciliter la résolution souhaitée.

8.2.3 Planification des méthodes de propagations

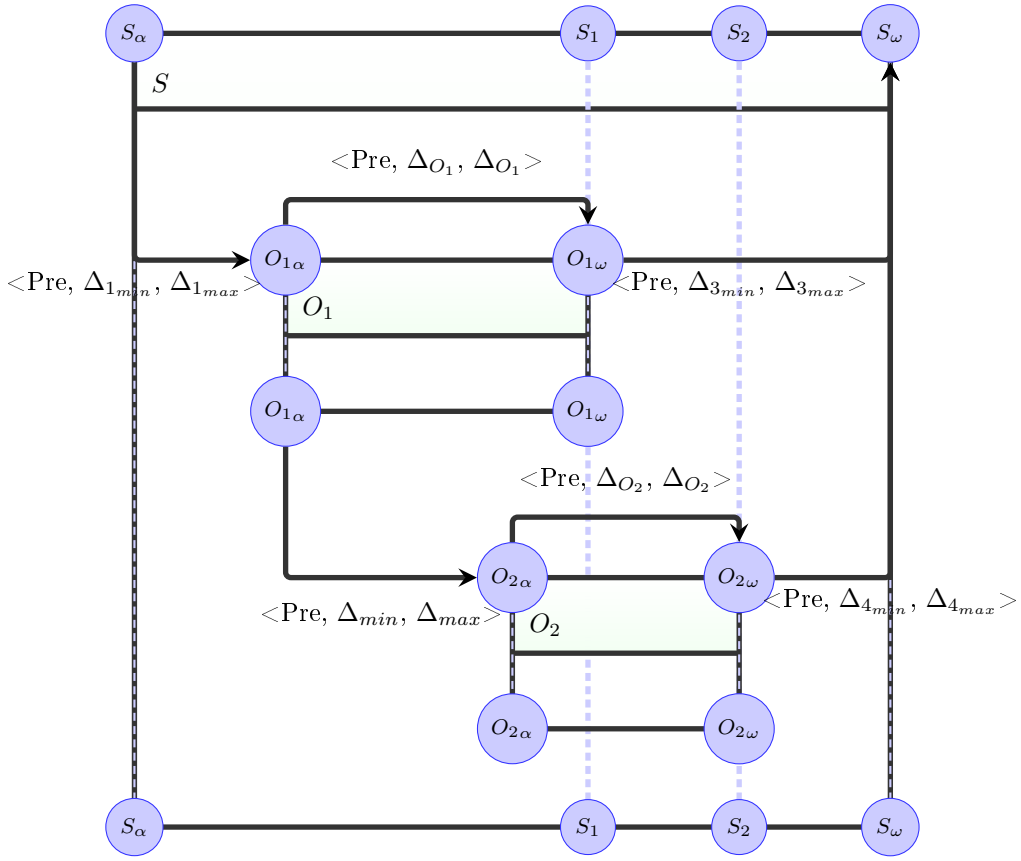
Une partie des choix de méthodes de propagation associées aux contraintes globales sont le fait du compositeur : contraintes d'intervalles, contraintes entre quanta, choix entre changement de tempo et changement de date... De cette planification par le compositeur peuvent naître des situations de conflit ou des cycles dans le graphe de contraintes.

La hiérarchie semble être la source de nombreuses situations de ce type. A titre d'exemple, nous présentons sur la figure 8.2, une situation dans laquelle une structure parent est définie avec la méthode de modification chronologique, tandis qu'une de ses structures enfant l'est avec la méthode anti-chronologique, des points de contrôle intermédiaires de cette structure enfant naît une situation de conflit. Sur cet exemple, une modification du point d'interaction Pi va déclencher une propagation chronologique sur les intervalles $\{\Delta_{O_3}\Delta_2\Delta_3\Delta_4\Delta_5\Delta_6\}$. Or ces intervalles sont liés avec ceux de la structure S. En effet :

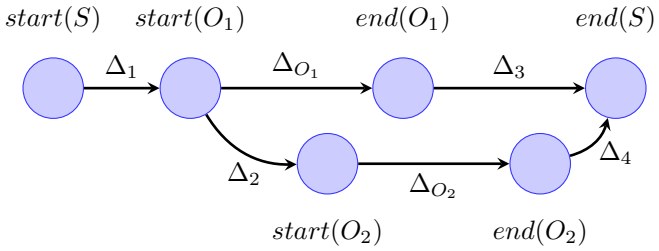
$$\Delta_3 = r(S).\Delta'_3$$
$$\Delta_4 = r(S).\Delta_{O_2}$$
$$\Delta_5 = r(S).\Delta_7$$

Par conséquent, la propagation chronologique au niveau de P va venir modifier les intervalles de S selon une stratégie différente de celle définie pour S.

L'aide à la composition doit être capable de guider le compositeur dans ses choix de méthodes et de détecter les situations de conflit. Ceci implique de maintenir un graphe de contrainte orienté pendant la phase d'édition.



(a) Un exemple de partition pour laquelle il est nécessaire d'identifier un point de contrôle pour exprimer des contraintes temporelles sous formes de relations. Ici le compositeur choisit de faire apparaître $end(O_1)$ et $end(O_2)$ au niveau des points de contrôle de S, or ces deux points sont liés par une contraintes temporelles sur l'intervalle les séparant. Cependant, l'expression de cette contrainte à l'aide de relations temporelles nécessite de faire apparaître un autre point de contrôle ($start(O_1)$) au niveau de S.



(b) Le graphe AOA associé à la partition ci-dessus

FIG. 8.1 – Une partition et son graphe AOA. La détection du point de contrôle à faire apparaître automatiquement au niveau de S peut se faire grace à un graphe AOA.

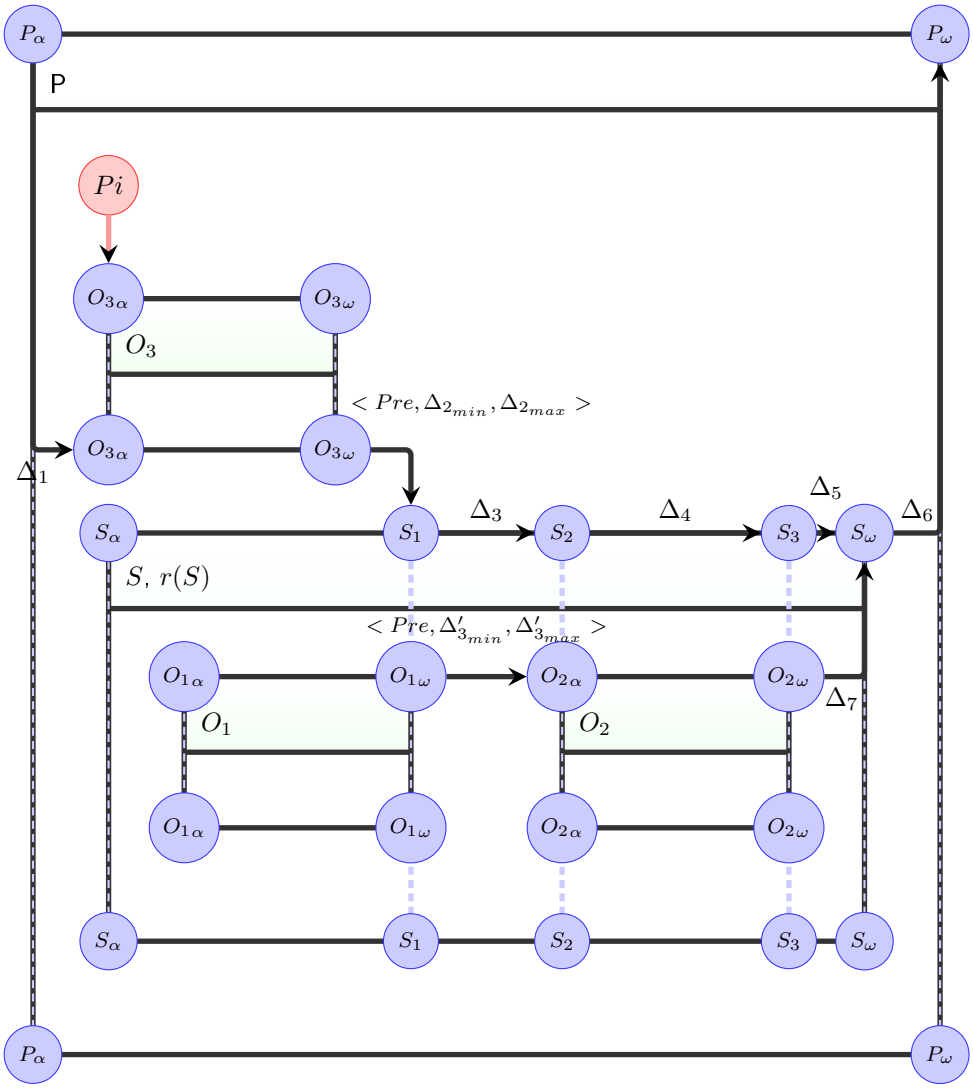


FIG. 8.2 – Un exemple de structure pouvant présenter une incompatibilité entre des methodes d'écrasements. Dans cet exemple nous supposons que la structure P est définie avec un comportement d'écrasement chronologique, tandis que la structure S est définie avec un comportement anti-chronologique. La représentation des points de contrôle de la structure S au niveau de P implique que les intervalles séparant ces points de contrôle prennent part à des contraintes d'intervalles au sein des 2 structures. Une modification au niveau du point d'interaction sur $start(O_3)$ va lancer une propagation chronologique sur les intervalles $\{\Delta_{O_3}\Delta_2\Delta_3\Delta_4\Delta_5\Delta_6\}$. Or $\Delta_3 = r(S).\Delta'_3$, $\Delta_4 = r(S).\Delta_{O_2}$,$\Delta_5 = r(S).\Delta_7$. Par conséquent, la propagation chronologique au niveau de P va modifier les intervalles de S selon une stratégie différente de celle définie pour cette structure.

8.2.4 Analyse de la concurrence

L'exécution des processus de manière concurrente peut naturellement conduire aux situations classiques de famine ou d'inter-blocages entre les processus. Dans son rôle d'analyseur de programmes, l'outil d'édition doit estimer ces risques et les signaler au compositeur.

8.2.5 Points d'interaction

Nous l'évoquions dans le chapitre concernant les partitions interactives, il est souhaitable que la définition des messages de déclenchement des points d'interaction ne conduise pas à introduire de synchronisation implicite. Cette situation peut survenir si deux points d'interaction déclenchables à un même moment sont associés au même message. Une solution simple pour résoudre ce problème est de détecter la contrainte temporelle entre les deux événements, éventuellement à l'aide du graphe AOA. Si une contrainte de précédence stricte existe entre les événements un même message pour les deux est possible, sinon le système l'interdit.

Une autre possibilité pour cette question est d'accepter l'indéterminisme lors de l'exécution. Si deux points d'interaction sont activables au meême moment avec le même message, à l'arrivée de ce message un seul sera déclenché sans que l'on puisse prédire lequel.

De plus, selon la stratégie de déclenchement des points d'interaction désirée par le compositeur, des limitations des plages temporelles d'activation des points d'interaction peuvent être nécessaires pour garantir la jouabilité de la partition.

Outre la complexité formelle que peut représenter la mise en œuvre de tous ces mécanismes de vérification, certains d'entre eux peuvent avoir des effets indésirables pendant la composition. En effet, il existe des exemples de pièces volontairement composées pour être "injouables" dans le but de forcer l'interprète (le contraindre) à faire des choix. Pour ce type de pièces, le contexte très sécurisé que nous venons de défnir ne semble guère adapté. Les décisions automatiques prises par le système doivent donc être modulables selon la volonté du compositeur.

Cependant, il est primordial que l'exécution de la partition en tant que système interactif soit assurée avant son interprétation, ce qui implique des mécanismes de vérfication quel que soit l'exigence et les choix du compositeur.

8.3 Machine d'exécution

L'interprétation d'une partition passe par l'implémentation du système interactif qu'elle constitue. Dans l'approche langage de programmation, cette étape est assimilable à une compilation de la partition décrite dans notre formalisme vers un code exécutable. Pour produire ce code, nous pourrions choisir de transformer directement la spécification de la partition dans un langage "courant" disposant d'un compilateur vers du code machine. La machine "générique" , valable pour toutes les partitions, que nous cherchons à définir, serait alors directement la machine physique ou éventuellement la machine virtuelle si le langage choisi en utilise une.

Cependant, il apparait clairement que plusieurs mécanismes se retrouvent dans toutes les partitions, comme par exemple la gestion des messages du musicien déclenchant les points d'interactions, le maintien des relations temporelles entre les événements, le calcul des modifications de dates par propagation... Un souci d'éfficacité plaide donc pour une mutualisaion du code associé à ces mécanismes communs. Cela permet en outre de se concentrer sur les propriétés propres de chacune des partitions lors de leur compilation.

8.3.1 Machine *ECO*

Ce choix impique la définition d'une machine abstraite effectuant les mécanismes communs en question, elle-même étant implémentée dans un code exécutable. Les partitions sont alors compilées en un code exécutable par cette machine et c'est l'exécution de ce code qui permet leur interprétation.

Présente dès le début de cette étude, cette idée nous a conduits à introduire la machine *ECO* dans [35].

Définition 8.1 *La machine ECO est une machine abstraite dont un état se définit comme un quadruplet :*

$$S =< E, C, O, t >$$

avec :

- E (Environment) *est un environnement musical*
- C (Controls) *est un flux d'événements datés fournis en entrée*
- O (Outputs) *est un flux de sortie*
- t *est la date absolue de l'état*

Dans cette définition, l'environement musical E contient tout ce dont la machine a besoin pour exécuter une partition. Nous distinguerons donc deux parties dans cet environnement, le *contenu temporel* qui permet à la machine de déterminer les dates auxquelles elle doit déclencher les processus de la partition, et les processus eux-mêmes, regroupés dans ce que nous appelons le *contenu procédural*. Le flux d'entrée C véhicule les actions discrètes du musicien tandis que O diffuse les résultats des processus destinés à l'environnement extérieur.

Le fonctionnement de la machine *ECO* est décrit par une suite de transitions entre des états successifs. Ces transitions sont synchronisées sur une horloge générale. L'exécution commence à la date 0, à laquelle la machine se trouve donc dans son état initial. Par la suite l'évolution de la machine se fait par transformation des éléments E, C et O à chaque tour de l'horloge générale.

Définition 8.2 *L'état initial S_0 de la machine est défini comme suit :*

$$S_0 =< E_0, C_0, O_0, 0 >$$

avec :

- *E_0 est une représentation de la partition interactive. Les dates d'occurrence des événements de la partition sont celles écrites dans la partition ; de la même manière, toutes les variables susceptibles d'évoluer au cours de l'exécution sont alors associées à leur valeur écrite.*
- $C_0 = \emptyset$
- $O_0 = \emptyset$

De plus, en notant δt le pas de l'horloge générale, on a :

$$< E, C, O, t > \rightarrow < E', C', O', t + \delta t >$$

avec :

- *E' est le résultat de la prise en compte des contrôles de C dont la date se situe entre t et $t + \delta t$. Si ceux-ci correspondent à des points d'interaction déclenchables entre t et $t + \delta t$, la valeur de date absolue des événements interactifs associés est fixée à $t + \delta t$. Ces valeurs sont propagées aux autres variables non encore fixées. Les événements dont la date absolue se trouve entre t et $t + \delta t$ sont déclenchés. Les modifications de valeur de variables temporelles dues à l'exécution de processus sont effectuées.*
- *C' contient la liste des contrôles discrets ayant une étiquette dont la date est comprise entre t et $t + \Delta t$*
- *O' contient les valeurs diffusées vers l'extérieur de la machine par les processus s'exécutannt à l'instant $t + \delta t$.*

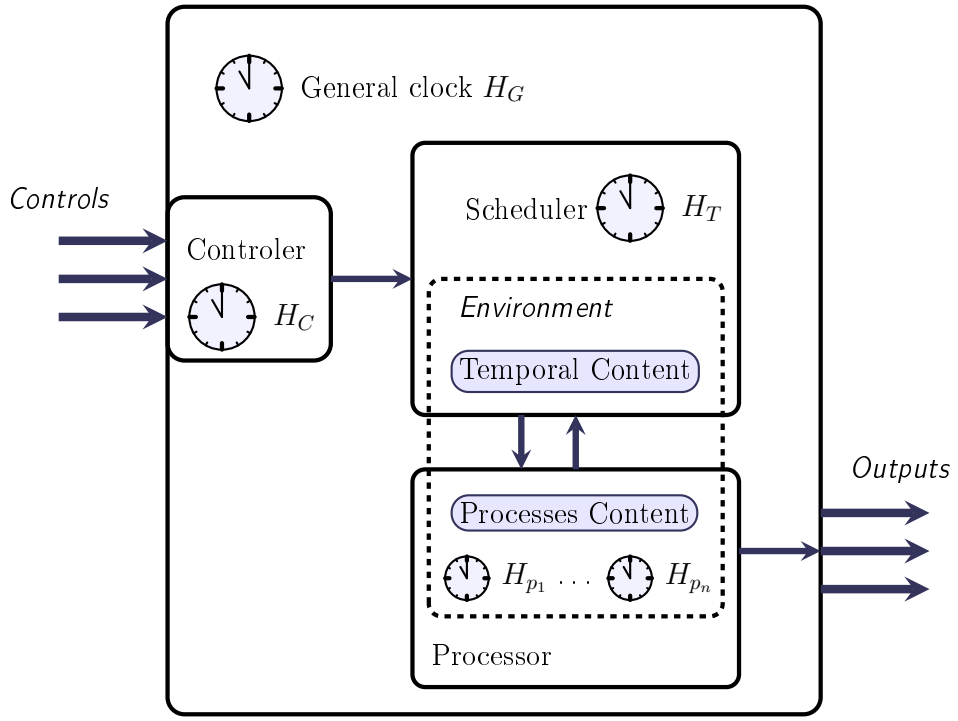


FIG. 8.3 – La machine ECO

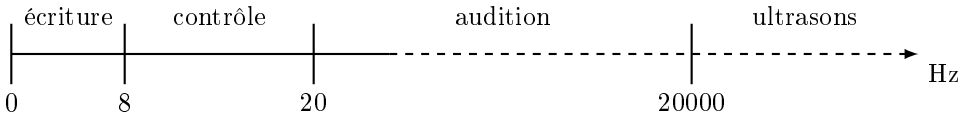


FIG. 8.4 – Echelle des fréquences de variation des paramètres

La figure 8.3.1 présente la structure de la machine *ECO*.

Les deux partie de l'environnement E sont manipulées par deux éléments distincts, l'*ordonnanceur* et le *processeur*. Une communication bi-directionnelle entre eux permet d'une part le déclenchement des étapes de calcul des processus lors du déclenchement de l'événement correspondant, et d'autre part, la modification de variables temporelles par l'exécution de processus. La figuration du *contrôleur* qui traite le flux C se justifie par la caractéristique de son horloge. De la même manière, nous avons présenté les horloges de chaque élément de la machine (H_G, H_C et H_T) et celles de chacun des processus (H_{p_i}). Les premières sont synchrones tandis que les secondes ne le sont pas nécessairement et dépendent des processus impliqués dans la partition et font donc partie de l'environement de la machine.

8.3.2 Cadence des horloges

Pour estimer les cadences utiles de ces différentes horloges, il nous faut analyser les paramètres manipulés par les éléments de la machine et les fréquences de variation de ces derniers. Desainte-Catherine et Marchand [37] proposent d'unifier la dualité musique/son sur une même échelle de fréquence de variation de paramètres. Cette échelle est présentée sur la figure 8.4, elle part des paramètres musicaux (macroscopiques) pour aller vers les paramètres sonores (microscopiques).

La première plage de fréquence correspond aux paramètres musicaux écrits par le compositeur : durée des notes, des mesures, phrasé .... La seconde plage correspond au contrôle des paramètres sonores

d'amplitude et de fréquence. La suite de l'échelle correspond aux paramètres sonores auditifs jusqu'à la valeur de 20000Hz à laquelle on passe dans le domaine des utltra-sons.

En partant de cette classification, on peut déduire qu'une cadence utile de l'horloge du contrôleur (H_C) est de l'ordre d'une vingtaine de Herz, la fréquence d'apparition des messages dans le flux C correspondant à des contrôles de paramètres de l'écriture (début et fins de note), mais aussi de certains paramètres sonores par l'entremise de points de contrôle intermédiaires.

Les cadences des horloges des processus sont fonction des paramètres manipulés par les processus. Elles peuvent monter jusqu'à plusieurs dizaines de milliers de Herz pour la synthèse sonore (paramètres auditifs), tandis qu'un processus modifiant un tempo sera cadencé à une fréquence de la plage des variations de paramètres d'écriture (inférieure à 8Hz).

Pour l'horloge H_T, il convient de s'intéresser à des notions de perception humaine du temps musical pour déterminer avec quelle précision temporelle les événements de la partition doivent être déclenchés. L'accuité temporelle dépend d'une part du contexte musical dans lequel on se place, et également de la position dans laquelle on se trouve. Lago et Kon [63] ont regroupé les résultats sur la question pour déterminer une latence acceptable pour les applications musicales. Deux cas se présentent selon si la personne dispose ou pas de retour haptique sur l'instrument (ou l'application) qui produit le son. Pour notre modèle la première situation correspond à un auditeur, un musicien se synchronisant avec les événements d'une partition interactive en cours d'exécution, ou encore à un musicien interprétant une partition interactive au moment du déclenchement d'événements statiques. Dans ce cas, la sensibilité est d'autant plus grande que la musique est rythmée, permettant dans certains cas particuliers la détection consciente de variations de 6ms sur l'intervalle séparant deux pulsations [1], voire une adaptation inconsciente à des variations de 4ms[89]. Dans un contexte moins rythmé, des délais entre des événements supposés synchrones pouvant aller jusqu'à 50ms sont observés dans la pratique instrumentale, sans que cela pertube l'écoute ou le jeu.

Dans le cas d'un retour haptique, c'est-à-dire en ce qui nous concerne, le déclenchement de points d'interaction en utilisant un périphérique haptique, en laissant de coté l'influence de la qualité du retour, [96] a montré que 20ms constituait une limite au-delà de laquelle un délai entre le geste et le déclenchement de l'événement devient perturbant, les délais inférieurs voyant le musicien s'adapter par anticipation.

Ces résultats indiquent qu'une cadence raisonnable pour H_T se situe en deçà de 1000Hz et qu'elle peut éventuellement descendre jusqu'à 50Hz. Vraissemblablement, une cadence adaptable au contexte de la partition à jouer et du dispositif de jeu semble idéale. L'horloge générale H_G trouvera certainement une cadence adéquate à 1000Hz.

8.3.3 Compilation des partitions

A présent qu'une machine d'exécution a été définie pour l'interprétation des partitions, il convient de s'intéresser à la compilation des partitions ecrites dans le formalisme à destination des compositeurs, vers un code exécutable par la machine *ECO*. Refléchir à la nature du code à produire, c'est réflechir aux opérations que doit effectuer la machine *ECO* au regard des fréquences d'horloge. Une fois identifiées les structures manipulés par la machine, le processus de compilation pourra être étudié.

Dans le cadre de la définition du code compilé et de son utilisation par la machine, nous garderons en tête que la quatlité d'une implémentation d'un langage est inversement proportionnelle au niveau de connaissance de l'implémentation interne nécessaire à un utilisateur pour produire du code utilisable. Ceci est d'autant plus critique dans notre cas que les compositeurs ne sont pas supposés disposer de connaissances d'informatique théorique. C'est d'ailleurs dans cette optique que nous avons limité les possibilités pour le compositeur de définir de nouvelles contraintes temporelles. Cette liberté nécessitant de connaître précisément les mécanismes de l'ordonanceur de la machine pour être réellement applicable.

Chapitre 9

Une implémentation partielle de la machine *ECO*

Nous présentons dans ce chapitre une implémentation de la machine *ECO* capable d'exécuter des partitions interactives écrites dans un formalisme simplifié. Cette première version répond à la problématique d'un modèle de partitions limité à un seul niveau de hiérarchie, avec un comportement de type point d'orgue pour tous les intervalles. La solution que nous apportons à ce modèle restreint précise l'architecture de la machine *ECO*.

En particulier, les solutions adoptées pour la gestion temporelle au sein de la machine, et les algorithmes permettant la compilation des partitions en un environnement exécutable, constituent le socle des solutions envisagées pour les modèles de partitions plus complets.

A la suite, nous présentons des modifications de cette solution de base visant l'utilisation des autres comportements d'intervalles. Sans pouvoir proposer une solution complète, nous donnons des éléments de solution pour le formalisme général des partitions.

9.1 Le formalisme du "point d'orgue"

Les partitions interactives que nous envisageons ici sont nettement plus simples que celles du modèle général. D'une certaine manière, on peut les considérer comme des partitions "de base", celles du formalisme complet étant des complexifications de ces partitions primaires. C'est d'ailleurs pourquoi la solution que nous proposons à ce cas simple sert de squelette aux solutions pour des modèles de partitions plus complets.

Les caractéristiques principales de ces partitions sont les suivantes :

- la hiérarchie est limitée à un seul niveau. Il n'y a qu'une seule structure, l'objet racine qui est de type *linéaire*. Tous les autres objets sont simples et donc des enfants directs de la racine.
- tous les intervalles sont *souples* i.e. :

$$\forall(\sigma_1, \sigma_2) \in \Sigma(racine)^2,$$
$$-\infty \leq date(\sigma_1, racine) - date(\sigma_2, racine) \leq \infty$$

- la structure racine est définie avec un *comportement* de type *point d'orgue*.
- la seule contrainte globale définissable est la limitation du nombre d'objets pouvnat se dérouler simultanément. Les objets sont *décalables* ou *mutables*, cette information est donnée par une fonction *statut* définie sur l'ensemble des textures de la partition.
- les processus sont simplifiés à l'extrême, considérés comme autonomes aucun branchement n'est possible entre eux. Ils sont supposés disposer d'arguments prédéfinis pour leurs calculs, ceux-ci ne présentant pas de contraintes intrinsèques.

On peut remarquer que la structure racine étant de type linéaire, seules des relations temporelles sont définissables entre les événements de la partition. De plus, tous les événements de la partition se produisent à chaque exécution puisqu'aucun choix n'est possible.

A partir de ces caractéristiques, il nous faut maintenant définir la représentation des partitions manipulées par la machine *ECO* et en déduire plus précisément la structure de cette dernière, ainsi qu'élaborer l'algorithme de compilation des partitions vers le code exécutable.

9.2 Compilation et exécution

Le choix des réseaux de Petri comme représentation des partitions manipulables par la machine *ECO* est le fruit de plusieurs réflexions. Tout d'abord, l'utilisation directe d'un système de résolution de contraintes impliquant les dates d'événements présente plusieurs inconvénients. Les ensembles de relations temporelles font très facilement apparaître des cycles dans le graphe de contraintes exigeant la mise en œuvre d'algorithmes de résolution gourmands. De plus, l'imbrication des variables de dates dans le réseau de contraintes rend malaisée l'estimation des conséquences d'une modification de date en terme de calcul. Un graphe de contraintes fortement connexe risque de propager la variation d'une date sur de nombreuses autres dates impliquant de nombreuses opérations. Or intuitivement, la propagation d'une variation de date n'est pertinente que sur quelques dates situées dans le futur proche de la date modifiée.

Il est donc important d'être capable d'orienter et de limiter les calculs nécessaires suite à une modification de date. En outre, un calcul est lancé lorsque l'indéterminisme sur la date d'un événement est levé par le déclenchement d'un point d'interaction, or nous disposons d'un ordre partiel entre les points d'interaction. Ces remarques suggèrent l'utilisation d'une structure permettant de séquencer des calculs locaux.

On l'a vu lors de la présentation de la théorie des réseaux de Petri, les propriétés de certains d'entre eux correspondent à celles que nous recherchons.

Les réseaux de Petri que nous utilisons ont les caractéristiques suivantes :

- **réseaux d'occurrences** : pour représenter l'ordre partiel entre les événements de la partition induit par les relations temporelles.
- **réseaux à flux temporels** : pour effectuer les calculs sur les dates lors des synchronisation au niveau des transitions avec plusieurs arcs entrants
- **réseaux synchronisés** : pour distinguer événements statiques et événements dynamiques, et conditionner le déclenchement de ces derniers à l'arrivée du contrôle du musicien

L'algorithme de compilation va consister à construire à partir d'une partition, un réseau de Petri représentant ces informations.

On peut également déduire que l'*ordonnanceur* de la machine *ECO* devra exécuter un réseau de Petri.

Rappelons que nous supposons ici que les partitions que nous manipulons sont *syntaxiquement* valides, c'est-à-dire qu'elles ne présentent pas d'incohérences temporelles et les valeurs des variables écrites respectent les contraintes.

9.2.1 Modélisation de l'ordre partiel

Pour modéliser l'ordre partiel des événements, nous nous appuyons sur les S-langages. La syntaxe et les propriétées ont été données dans le chapitre 3.

Rappelons en préliminaire une propriété du pouvoir d'expression de l'opération de jointure dans les S-langages : la jointure de deux S-langages S_1 et S_2 représente l'ensemble des S-mots vérifiant à la fois les contraintes représentées par S_1 et les contraintes représentées par S_2.

Par conséquent, soient P une partition, Σ_P l'ensemble de ses événements et $R = \{rt_i, i \in [|1, n|]\}$ les relations temporelles entre ses événements, on note $\mathscr{L}(P)$ le S-langage modélisant l'ordre partiel entre les événements de P.

Comme tous les événements de P doivent se produire, alors le S-langage de la partition sans contraintes est $\mathscr{U}(\widehat{\Sigma_P}, (1, \ldots, 1))$, et

$$\mathscr{L}(P) = \mathscr{U}(\widehat{\Sigma_P}, (1, \ldots, 1)) \underset{r \in R}{\bowtie} s_r$$

où s_r est le S-mot de $\widehat{\Sigma_P}^*$ représentant l'ordre imposé par r.

Cependant, étant donné les caractéristiques de R : pas de conflits et présence des relations de cohérence des objets qui assurent un ordre total entre le début et la fin de la partition et les points de contrôle d'un même objet, alors :

$$\mathscr{U}(\widehat{\Sigma}_P, (1, \ldots, 1)) \underset{r \in R}{\bowtie} s_r = \mathscr{U}_{|1|} \underset{r \in R}{\bowtie} s_r$$

avec $\mathscr{U}_{|1|}$ défini sur $\widehat{\Sigma}$.

Comme R ne contient que des synchronisations et des mises en précédence, $\mathscr{P}$ est un S-langage de réseau d'occurrences.

La compilation P en code exécutable par la machine *ECO* consiste donc dans un premier temps à construire le réseau d'occurrences représentant le S-langage de P.

Étant donné la construction du réseau présentée dans le chapitre précédent, dorénavant nous identifierons indifféremment une transition du réseau représentant la partition et les événements de Σ_P qu'elle représente.

Quelques remarques concernant la structure du réseau, tout d'abord, l'absence de conflits implique qu'une place du réseau est suivie d'une seule transition. De plus, étant donné l'ensemble de contrainte et la construction, le support du marquage initial du réseau est la place précédant la transition associée au début de la partition. En outre, le marquage terminal à définir pour avoir un langage de réseau égal au S-langage de P, a pour support la place succédant à la transition de fin de la partition.

9.2.2 Modélisation des contraintes temporelles quantitatives

Pour intégrer du temps dans les réseaux d'occurrences, il faut définir la sémantique des transitions et la fonction de temporisation des arcs des réseaux à flux temporels.

Définition 9.1 *Soient P une partition interactive, PN le réseau modélisant P, nous définissons les fonctions Sem et Ita de PN de la manière suivante :*

- Sem : $\forall t \in T, Sem(t) = Et - Pur$
- ita : $\forall p \in P / p_{start(racine)_1}, ita(p, p^\bullet) =$
  $(0, date(racine, p^\bullet) - date(racine, {}^\bullet p), \infty)$
  $ita(p_{start(racine)_1}, t_{start(racine)}) = (0, 0, \infty)$

Rappelons que les valeurs temporelles présentes sur un arc sont : une date de tir au plus tôt, une valeur nominale et une date de tir au plus tard, et que la sémantique du *Et-Pur* stipule qu'une transition ne peut être tirée que durant l'intervalle intersection des intervalles de ses arcs entrants. Le choix de sémantique traduit donc le respect absolu des relations temporelles.

Nous venons de le voir, une place représente le temps d'attente entre l'ensemble d'événements représenté par sa transition "prédécesseur" et l'ensemble d'événements représenté par sa transition "successeur". Par conséquent la valeur nominale de ce temps d'attente est bien la différence des dates écrites dans la partition. En outre comme tous les intervalles sont souples, toutes les valeurs de $\mathbb{R}_+$ sont acceptables comme valeur d'intervalle à l'exécution de la partition.

Ainsi, les dates de tir au plus tôt et au plus tard de chaque transition sont bien 0 et $+\infty$.

Comme les valeurs de dates écrites dans la partition sont valides pour les contraintes, les dates d'événements synchronisés, et donc représentés par la même transition dans le réseau, sont les mêmes. Par conséquent, la définition de la date de tir nominale est valide, même si plusieurs événements sont représentés par la transition prédécesseur ou la transition successeur d'une place p.

Concernant le début de la partition, l'arc entrant de la transition représentant cet événement ne peut avoir de valeur nominale par définition (du point de vue de la partition, le temps n'est pas défini avant son début). Une valeur nulle permet de maintenir pragmatiquement la place prédécesseur de la transition de début dans le rôle de seule place marquée à l'état initial, sans ajouter de signification temporelle. Les valeurs de tirs au plus tôt et au plus tard, outre leur homogénéité avec les autres sont utiles dans le cas interactif comme nous le verrons bientôt.

On peut légitimement s'interroger sur la suppression des places implicites lors de la construction du réseau d'occurrences et se demander si les informations temporelles qu'elles portent ne sont pas essentielles. Rappelons que ces places formalisées dans la section 4.2.5.0 n'ont pas d'incidence sur l'exécution d'un réseau de Petri non temporisé.

En fait comme les partitions sont cohérentes, la valeur de tir nominale d'un arc sortant d'une place implicite est égale à la somme des valeurs nominales des arcs sur le chemin rendant la place implicite. De plus comme tous les intervalles sont souples, les bornes, valeur de tir au plus tôt et au plus tard de l'arc sont sans intérêt. On peut les supprimer sans perdre d'informations temporelles.

9.2.3 Les points d'interaction

Le conditionnement du franchissement des transitions au déclenchement de points d'interaction s'appuie sur le formalisme des réseaux de Petri synchronisés exposé dans la section 3.6.

Pour prendre en compte les points d'interaction, il nous faut définir la fonction $Sync$ du réseau PN.

Définition 9.2 *Soient P une partition interactive, PN le réseau représentant P et $\mathscr{P} = \{pi_k =< \sigma_{i_k}, p_k >\}_{k\in[|1,n|]}$, l'ensemble des points d'interactions de P.*

$$\begin{array}{ll} & Sync(start(P)) = e_{n+1} \\ On\ a : & \forall \sigma \in \Sigma_P \setminus start(P)\ t.q.\ \exists pi_k =< \sigma_{i_k}, p_k >, Sync(\sigma) = e_k \\ & Sinon\ Sync(\sigma) = e_0 \end{array}$$

Chaque point d'interaction est ainsi représenté par un unique événement dans le réseau.

Cette définition permet le respect de l'ordre partiel entre les événements quels que soient les tentatives de déclenchement du musicien lors de l'exécution. En effet, étant donnée la règle de tir des réseaux de Petri synchronisés, si le franchissement d'une transition interactive σ_i est conditionné par l'arrivée d'un événement e_i, la prise en compte de e_i n'est possible que si σ_i est sensibilisée. Or cette sensibilisation n'intervient que si toutes transitions devant précéder t ont été franchies.

Au cours de l'exécution, cela signifie que le déclenchement d'un point d'interaction susceptible d'invalider l'ordre partiel entre les événements de la partition ne sera pas pris en compte.

Enfin, rendre la transition de début de partition interactive permet de déclencher interactivement le déroulement de la partition.

9.2.4 Choix des dates de tir des transitions

Dans le formalisme des réseaux de Petri à flux autonomes, la sémantique d'une transition permet de définir les bornes de l'intervalle pendant lequel la transition *peut* être tirée. Il convient donc de choisir une date de tir pour chaque transition dans son intervalle de validité. Dans notre cas, comme tous les arcs menant à une transition ont pour intervalle $[0, \infty]$, la sémantique *Et-Pur* donne pour toutes les transitions l'intervalle de validité suivant :

$$[\max_{1\leq i\leq n}(\tau_i), \infty]$$

avec les $\{\tau_i\}_{i\in[|1,n|]}$ les dates d'activation des arcs entrants dans la transition. Ce qui correspond dans ce cas particulier où tous les intervalles de la partition sont souples, à imposer uniquement l'ordre partiel entre les événements.

Pour les transitions interactives, la date choisie est celle du déclenchement de leur point d'interaction.

Pour les transitions statiques, le calcul de cette date dépend du comportement que l'on souhaite définir, ici le comportement *point d'orgue* présenté dans la section 7.3.2.

Rappelons que dans cette sémantique la répercussion de la modification d'un intervalle se fait au niveau des points de synchronisation qui suivent l'événement modifié. Cependant, d'autres modifications peuvent intervenir avant d'atteindre le premier point de synchronisation. C'est donc au niveau des transitions reflétant ces points de synchronisation que les calculs doivent s'effectuer.

Au niveau de ces points, le comportement *point d'orgue* cherche à respecter la valeur d'un intervalle sauf si celle-ci doit être augmentée pour respecter un retard intervenu dans un autre chemin du réseau de Petri. Nous en déduisons le calcul de date pour une transition statique.

Soient σ une transition statique du réseau et $A_\sigma = \{a_i\}_{i\in[!1,n!]}$ les arcs entrants dans σ. Soit $i \in [|1,n|]$ on note $ita(a_i) =< \alpha_i, n_i, \beta_i >$ et τ_i la date d'activation de a_i alors la date de tir τ_σ de σ s'exprime par :

$$\tau_\sigma = \max_{1\leq i\leq n} (\tau_i + n_i)$$

Étant donné l'intervalle de validité infini de la transition, ce choix est valable; de plus ce choix correspond bien à la définition du *point d'orgue* au niveau des synchronisations puisque les intervalles seront conservés ou agrandis. Pour le cas particulier des transitions n'ayant qu'un seul prédécesseur, cette définition est également valable.

Accessibilité du marquage final

Le lecteur attentif a en mémoire une propriété importante des réseaux de Petri à flux autonomes : la question de l'accessibilité est indécidable pour ce type de réseau dans le cas général. Or, pour reprendre le vocabulaire des partitions interactives, la propriété de *jouabilité* d'une partition se traduit au niveau du réseau de Petri qui la modélise, par la question de l'accessibilité du marquage terminal à partir de l'état courant du réseau. En effet, l'état du réseau traduit l'exécution partielle de la partition.

Dans notre cas, comme l'intervalle de validité de chaque transition est égal à $[0,\infty]$, n'importe quelle date de tir est acceptable pour une transition donnée et permet donc d'accéder au marquage final. En particulier, les choix du musicien pour les transitions interactives et les calculs de date pour les transitions statiques, permettent l'accessibilité du marquage final. Par conséquent la *jouabilité* de la partition est assurée quels que soient les choix de date du musicien.

9.2.5 Contraintes Globales

Pour prendre en compte les contraintes globales, il est nécessaire de disposer d'un mécanisme permettant de répondre à la requête : "Étant donné l'état courant des variables non temporelles de ma partition, le déclenchement de l'objet O est-il possible?".

Dans notre contexte de processus concurrents, ce type de mécanisme est typiquement ce que proposent les formalismes de programmation concurrente par contraintes comme *NTCC*. Nous nous inspirons donc des solutions apportées par ce formalisme en utilisant un *magasin de contraintes*.

Dans le contexte de la programmation concurrente par contraintes, un *magasin de contraintes* est une structure permettant d'accumuler des connaissances sur un ensemble de variables. Les processus concurrents peuvent ajouter des connaissances dans le magasin grâce à une fonction classiquement notée *tell* ainsi que l'interroger, pour savoir si les informations qu'il contient à un instant donné permettent de déduire une information sous forme de contrainte (déterminer si la contrainte est vérifiée ou non), grâce à une fonction *ask*. Éventuellement, le magasin n'a pas suffisamment d'informations pour répondre et dans ce cas le processus est bloqué jusqu'à réception d'une réponse. Selon les réponses que fournit le magasin, les processus peuvent être lancés, retardés, modifiés ... Les informations globales du *magasin* permettent une communication entre les processus.

La ressemblance forte de ces mécanismes avec les caractéristiques de notre modèle nous a poussé à envisager l'utilisation de *NTCC* pour implémenter la machine *ECO*, une ébauche d'étude à ce sujet est présentée dans [7]. Cependant, le mélange de toutes les contraintes temporelles et non-temporelles dans un magasin de contraintes conduit aux risques de calcul trop long évoqués plus haut. Séparer les deux types de contraintes dans deux structures différentes, permet de donner facilement la priorité aux variables temporelles. Dans notre cas c'est donc l'*ordonnanceur* exécutant le réseau de Petri, qui au moment de franchir une transition de début d'objet interroge le *magasin de contraintes*.

Notre formalisme simplifié ne considérant que les contraintes sur le nombre maximum d'objets exécutables simultanément, le magasin de contraintes manipule donc les variables booléennes v_{ex} , les variables $nb - tex$ et $nb - tex_{max}$ de la structure racine. La nature de ces contraintes assure que le magasin est toujours capable de déterminer si une transition est franchissable, connaissant le nombre d'objets qu'elle déclenche.

Si la réponse est positive, la transition est franchie dans le cas contraire, la stratégie dépend du statut de l'objet. Si l'objet est *mutable*, la transition est franchie, le processus est exécuté, mais ses sorties ne produisent pas de valeurs.

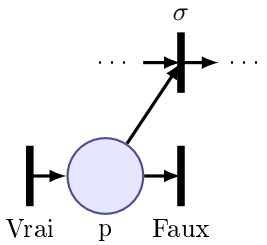


FIG. 9.1 – Place précédant la transition de début d'un objet décalable

Si l'objet est *décalable*, alors la transition ne doit pas être franchie tant que le *magasin de contraintes* ne répond pas favorablement à la requête.

Il y a deux possibilités pour modéliser ce mécanisme dans le réseau de Petri. D'une part, ajouter une place précédant chaque transition représentant le début d'un objet décalable, la présence d'un jeton dans cette place étant conditionnée par la possibilité de déclencher un objet. La figure 9.1 présente la configuration rendant possible cette solution.

Le franchissement des transitions *vrai* et *faux* est conditionné à la possibilité de déclencher un objet (au travers d'événements de réseaux synchronisés).

Plus simplement, on peut utiliser les réseaux de Petri à prédicats qu'on trouve dans *DoubleTalk* de Pope [80], et qui permettent de conditionner le franchissement d'une transition à la véracité d'un prédicat. Nous conditionnons alors le déclenchement des transitions de début d'objets décalables avec la véracité de la requête *ask* sur le magasin de contraintes.

9.2.6 Algorithme de compilation

La compilation d'une partition en réseau de Petri suit exactement la présentation que nous venons de faire, elle comprend donc les étapes suivantes :

- création du réseau d'occurrences
- ajout des dates de tir sur les arcs
- définition des événements de réseaux synchronisés
- ajout des prédicats sur les transitions de début d'objets décalables

Comme une partition est finie, l'algorithme termine. De plus, étant données les propriétés des réseaux d'occurrences démontrées précédeemment, ainsi que les précisions apportées ici, le réseau de Petri modélise bien les propriétés de la partition.

L'exécution de la partition consiste à exécuter le réseau de Petri avec l'aide d'un magasin de contraintes.

9.3 Machine ECO

La structure du code exécutable par la machine *ECO* nous permet de définir plus précisément l'architecture de la machine.

L'ordonnanceur est constitué d'un module d'exécution de réseau de Petri à flux temporels et synchronisés, tandis que le processeur contient les processus et le magasin de contraintes. La figure 9.2 présente l'architecture de la machine *ECO*.

Le fonctionnement de la machine se fait de la manière suivante à chaque tour d'horloge :

- le **contrôleur** récupère les messages extérieurs et les fait passer à l'ordonnanceur sous forme d'événements de réseaux synchronisés
- l'**ordonnanceur** récupère les événements de réseaux et les date par rapport à sa propre horloge H_T, puis exécute le réseau : prend en compte les événements extérieurs attendus, franchit les transitions franchissables en interrogeant(*ask*) le magasin de contraintes, et communique au processeur les étapes de processus à effectuer.

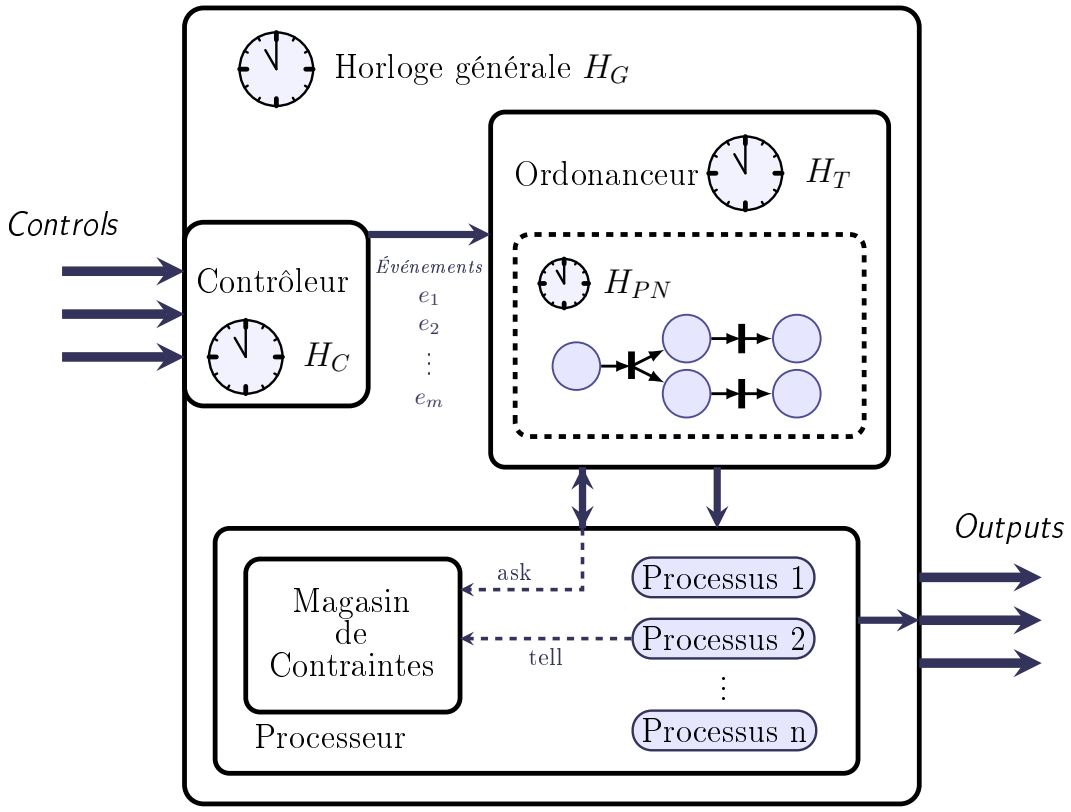


FIG. 9.2 – Architecture de la machine *ECO* pour le formalisme du "point d'orgue"

– le **processeur** effectue les étapes de calcul requises des processus, dirige la sortie des processus vers la sortie de la machine et met à jour le magasin de contraintes (*tell*).

Horloges

Au cours de l'exécution du réseau de Petri, le calcul des bornes de l'intervalle de validité d'une transition et sa date de tir dépendent des dates d'activation des arcs entrants, donc de l'horloge cadençant l'exécution du réseau de Petri H_{PN}. Cette horloge exprime les dates des événements dans le référentiel temporel de la structure racine et en reprenant la notation de franchissement d'une transition on a :

$$\tau_\sigma = date(e_\sigma, racine)$$

avec e_σ n'importe quel événement représenté par σ.

En outre, l'horloge H_T de l'ordonnanceur correspond au référentiel temporel absolu.

Par conséquent, le ratio $r(racine)$ entre le quantum q_{racine} et le quantum q_{abs} peut être simulé en introduisant le même rapport entre le pas des deux horloges.

Disposant ainsi du rapport $r(racine)$ lors de l'exécution de la partition, les changements de tempo sont aisés à mettre en œuvre.

Soient σ une transition interactive du réseau, si le point d'interaction lié à cette transition modifie le tempo, alors le franchissement de σ est suivi d'une mise à jour de $r(racine)$ qui s'exprime de la manière suivante :

$$r(racine) \leftarrow \frac{\max_i(\tau_i + n_i)}{date(\sigma)}$$

où $date(\sigma)$ est la date absolue du franchissement de σ (i.e. la date d'arrivée de l' événement de réseau synchronisé qui conditionne ce franchissement) exprimée dans le référentiel de l'horloge H_T, et $\max_i(\tau_i + n_i)$, la date "attendue" dans le réseau pour ce franchissement exprimée dans le référentiel de H_{PN}. Ce calcul correspond exactement à la propagation de la modification de la date absolue d'un événement vers le tempo de sa structure parent.

9.4 Formalisme avec rigidité des intervalles

A partir du formalisme de base et des solutions que nous y apportons, nous construisons une implémentation de la machine *ECO* pour une version du formalisme dans laquelle la rigidité des intervalles est acceptée.

Les partitions auxquelles nous nous intéressons ici sont donc la restriction des partitions générales à un seul niveau hiérarchique, la structure racine étant de type linéaire. Les bornes des intervalles associés aux relations temporelles peuvent être quelconques, et un comportement est défini pour la structure racine parmi :

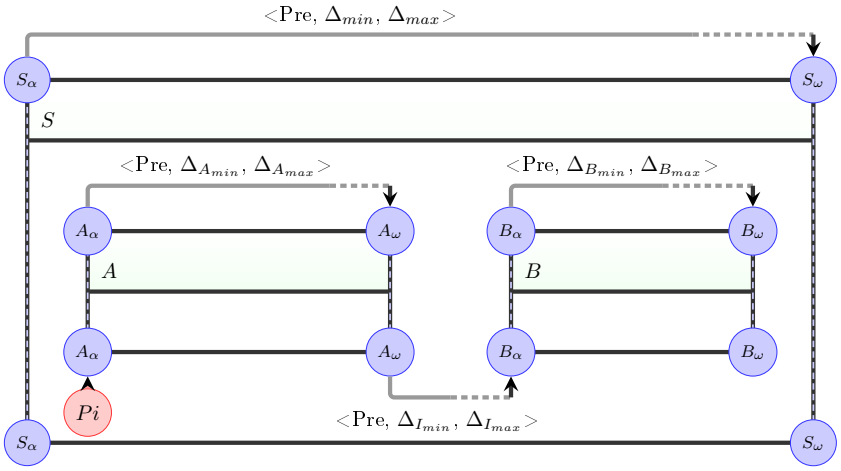
– point d'orgue
– modification chronologique
– modification anti-chronologique
– modification proportionnelle

Dans ce contexte, la notion de *jouabilité* prend toute sa dimension, puisque certains choix de date d'événements interactifs peuvent conduire à des incohérences temporelles.

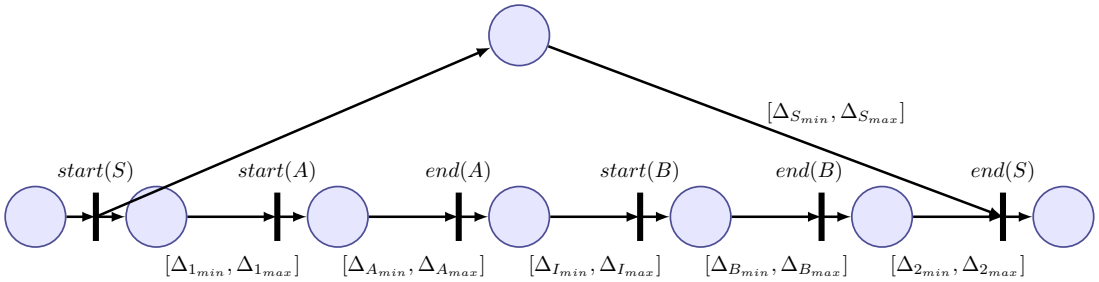
Nous supposerons ici que la *jouabilité* des partitions est assurée étant donnée les valeurs de variables temporelles de la partition écrite.

Pour exécuter ce type de partitions, deux problèmes supplémentaires apparaissent par rapport au cas précédent :

– respecter les intervalles de validité des points d'interactions
– modifier les valeurs des variables temporelles en fonction des déclenchements opérés par le musicien et du comportement de la structure racine



(a) Un exemple de structure



(b) Le réseau de Petri associé. Pour l'obtenir, la compilation est effectuéde la même manière que le cas où tous les intervalles sont souples. Les intervalles associés au relations temporelles sont associés aux arcs représentant ces relations. Enfin les places précédemment considérées commpe implicites sont conservées.

FIG. 9.3 – Un exemple de structure ainsi qu'un réseau de Petri la modélisant.

9.4.1 Respect des intervalles de validité

La résolution du premier point s'intègre très facilement à la solution basée sur les réseau de Petri, tandis que la seconde si elle s'envisage aisément dans le cas du comportement point d'orgue, dépasse les capacités d'un réseau de Petri dans les autres cas.

En reprenant les situations d'exemple simple de modifications, on peut se rendre compte de ce que nous avançons ici. La figure 9.3 présente le cas d'une partition dont on supposera qu'elle définit avec un comportement point d'orgue. Nous présentons également un réseau de Petri permettant de modéliser cette structure tout en permettant de vérifier le respect des bornes d'intervalles. Ce réseau a les mêmes caractéristiques que ceux présentés précédemment (flux temporels, sémantique du *Et-Pur*...).

Ce réseau est obtenu par une compilation similaire à celle utilisée précédemment pour le cas où tous les intervalles sont souples. Les arcs du réseau précédant une transition portent des valeurs de tir correspondant aux propriétés des relations temporelles dont ils sont les représentations (i.e. les dates de tir au plus tôt et au plus tard sont les valeurs extrêmes de l'intervalle de la relation, plus nécessairement 0 et ∞). Le choix de la date de tir d'une transition reste le même.

On peut observer, que nous avons volontairement laissé dans le réseau une place que nous considérions précédemment comme implicite. Celle-ci modélise en effet une information sur l'intervalle de temps qu'elle porte (Δ_S), et participe de ce fait aux calculs locaux des bornes des intervalles de tir des transitions.

Cet exemple permet d'illustrer deux situations différentes face au respect des intervalles de validité.

Le cas de l'intervalle Δ_1 précédant le point d'interaction se présente comme un problème de respect direct d'un intervalle de validité. En effet, étant données les règles de tir des réseaux à flux temporels, l'activation du point d'interaction ne sera possible qu'après la date de tir au plus tôt. A l'arrivée de la

date de tir au plus tard, si le point d'interaction n'a pas été déclenché, alors le système risque de se trouver face à une violation de l'intervalle de validité de Δ_1.

Comme nous l'avions évoqué dans la partie concernant le formalisme des partitions, plusieurs stratégies sont ici possibles, en fonction de la volonté d'assurer ou non la *jouabilité* de la partition :

- le système peut franchir automatiquement la transition pour assurer la *jouabilité*
- si celle-ci n'a pas à être assurée, le système peut signaler cette violation et arrêter l'exécution, ou la poursuivre si le compositeur admet le non respect de certaines contraintes.

Il est important de noter que la première solution n'est acceptable ici que parce que le franchissement de la transition est possible compte tenu des règles de tir des réseaux à flux temporels. Or ce n'est pas nécessairement le cas, et un choix de date inappropriée pour un point d'interaction peut conduire à la violation d'une contrainte temporelle en aval de ce point d'interaction, rendant impossible le franchissement automatique de la transition considérée.

Sur l'exemple de la figure 9.3, la transition $end(S)$ permet d'illustrer cette situation. Une valeur trop importante de Δ_1, peut conduire à ce que $\Delta_{S_{max}}$ expire avant que $\Delta_{2_{min}}$ ne se soit écoulé, rendant ainsi impossible le franchissement automatique de $end(S)$ pour rester dans l'intervalle de validité de Δ_S.

Ainsi, l'approche nouvelle des places "implicites" permet d'obtenir une représentation des partitions dans laquelle les intervalles de validité des points d'interaction sont soit respectés, soit leur violation est détectée.

Cependant, assurer la jouabilité dans ces conditions nécessite une analyse plus fine des partitions pour les transformer en réseau. Nous discutons ce point un peu plus loin.

9.4.2 Modification des valeurs d'intervalles et comportement

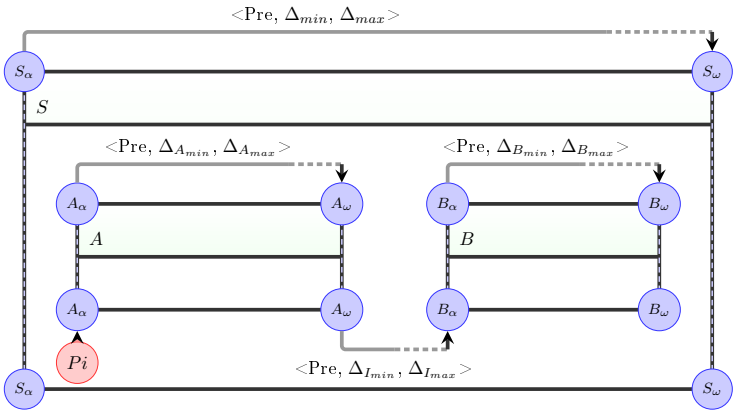
La réaction du système aux déclenchements des points d'interaction en fonction du comportement est un problème nettement plus épineux. En effet, son intégration directe dans le réseau de Petri n'est pas possible dans le cas général. On l'a vu, les réseaux de Petri à flux temporels simulent bien le comportement point d'orgue. Cependant les autres comportements nécessitent quelques aménagements. Nous présentons ici une piste d'aménagement que nous avons explorée pour le comportement anti-chronologique.

Sur la figure 9.4, nous présentons une structure définie avec un écrasement anti-chronologique et sa modélisation par un réseau de Petri coloré avec compteurs.

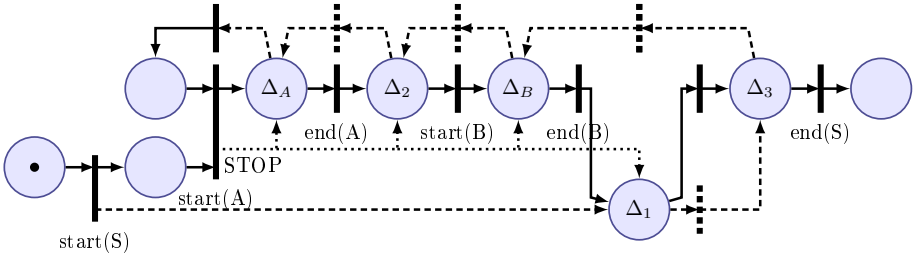
On trouvera une description du formalisme des réseaux de Petri colorés dans [61]. Un réseau de Petri coloré permet d'affecter des couleurs aux arcs et aux jetons du réseau. Naturellement, les couleurs des arcs sont liées à celles des jetons et les règles de tir des transitions sont modifiées. Soient une transition t et une place p qui précède t avec a un arc de couleur c reliant p à t. Alors pour que t soit sensibilisée, il faut qu'un jeton de couleur c soit présent dans p. De la même manière, lors du franchissement d'une transition t, le jeton produit dans une place p succédant à t est de la couleur de l'arc reliant t à p. L'utilisation de plusieurs couleurs dans le réseau nous permet de séparer les arcs et jetons d'évolution du réseau (couleur noire) des arcs et jetons permettant la modification des valeurs d'intervalles séparant les événements en fonction du déclenchement des points d'interaction (couleur verte). Les valeurs d'intervalles associées aux arcs du réseaux sont sont modifiables par des compteurs déclenchées par la production d'un jeton vert dans les places. La valeur initiale du compteur est la valeur nominale de l'intervalle, elle décrémentée jusqu'à 0. Lorsque 0 est atteint, la transition qui suit la place par un arc vert peut être tirée. L'enchaînement des arcs verts suit l'ordre anti-chronologique et permet ainsi de décrémenter les compteurs associés aux intervalles dans cet ordre. Un déclenchement du point d'interaction provoque l'arrêt des compteurs et le maintien de leur valeur pour les arcs noirs et ainsi un calcul des nouvelles valeurs des intervalles.

La solution particulière que nous venons d'exposer rend la compilation des partitions très lourde par la multiplication des cas particuliers. De plus, elle n'est pas transposable aux autres types de modifications. Enfin, et c'est là l'argument principal de son abandon, elle nécessite d'identifier au sein de la partition les combinaisons linéaires correspondant aux contraintes sur les intervalles. Les ajouts d'arcs ou de places avec compteur permettant alors de calculer de manière "continue" les nouvelles valeurs des variables impliquées dans ces combinaisons linéaires.

Devant ces obstacles, nous avons totalement abandonné cette solution et choisi de nous orienter vers un système hybride.



(a) L'exemple simple d'une structure définie avec un comportement anti-chronologique



(b) Un réseau de Petri coloré permettant l'écrasement anti-chronologique de la structure ci-dessus. Sur cette figure, trois couleurs d'arcs sont représentées. La couleur "noire" est représentée en traits pleins, la couleur "verte" est représentée avec des tirets et la couleur "rouge" est représentée en pointillée. Dans les réseaux de Petri colorés les jetons sont également colorés, un jeton produit dans une place est de la couleur de l'arc entrant qui conduit à la production du jeton. La couleur noire est celle du déroulement de la structure. La couleur verte permet de calculer les valeurs d'intervalles en continu pendant le déroulement de la structure. La production d'un jeton vert dans une place déclenche le décompte d'un compteur dont la valeur initiale est la valeur inscrite dans la place. lorsque le compteur est nul, la transition suivant la place est franchie. Les arcs rouges permettent d'arrêter les compteurs des places. Le franchissement de la transition $start(A)$ dépend de la présence d'un jeton dans l'une ou l'autre des places qui la précèdent. Son franchissement depuis la place du bas étant conditionné au déclenchement du point de contrôle. Le fonctionnement du réseau de Petri est le suivant : au déclenchement de la structure, un jeton est créé dans la place précédant la transition $start(A)$, et celle-ci peut être franchie par le déclenchement de Pi. Un jeton vert est créé dans la place Δ_1 dont le compteur est décrémenté. Si le déclenchement de Pi est très retardé, les intervalles vont être réduits dans l'ordre formés par les arcs "verts". Au déclenchement de Pi, la valeur courante de chaque intervalle est figée, ces valeurs étant celles utilisées pendant l'exécution du réseau. Si Pi est anticipé par rapport à ce qui est écrit, le temps restant de Δ_1 est ajouté à $Delta_3$ par la présence de la place Δ_1 avant Δ_3 dans l'ordre des arcs noires. Ces calculs des intervalles correspondent bien à l'écrasement anti-chronologique. Dans le cas extrême où les intervalles sont totalement réduits, un jeton noire est créé dans la place précédant la transition $start(A)$. Ceci provoque le déclenchement de la transition et le déroulement du réseau avec des valeurs d'intervalles nulles.

FIG. 9.4 – Une utilisation d'un réseau de Petri coloré avec compteurs pour modéliser le comportement d'écrasement anti-chronologique

9.4.3 Solution générale pour le formalisme avec rigidité des intervalles

Quitte à déterminer ces combinaisons linéaires, autant utiliser les formules de propagation de l'annexe A, et effectuer le calcul à partir du moment où on cannait la date de déclenchement du point d'interaction. Cette approche a le bon goût d'être valable pour tous les comportements.

Cependant, il est à noter qu'une telle solution implique la mise en place d'un algorithme de propagation au travers d'un graphe de contraintes, et il est tout à fait légitime de s'interroger sur la cohabitation d'une telle structure avec un réseau de Petri, notamment vis à vis des redondances qu'une telle cohabitation peut générer.

Mais les contraintes du réseau de propagation ayant pour but de permettre au système de réagir au déclenchement des points d'interaction, il est possible d'identifier avant l'exécution de la partition quelles contraintes vont être sollicitées par les déclenchements de points d'interaction, et ainsi de ne conserver dans le graphe de contraintes que celles qui ne sont pas redondantes avec le réseau de Petri.

Ayant retenu cette solution, nous nous proposons pour aboutir au graphe de contraintes d'adopter la démarche suivante :

- construire le réseau de Petri représentant les contraintes temporelles de la partition
- identifier les combinaisons linéaires correspondant aux contraintes d'intervalles en analysant le réseau de Petri
- construire un graphe de contraintes général contenant toutes les contraintes d'intervalles identifiables
- analyser l'emplacement des événements dynamiques dans le réseau de Petri en relation avec le comportement de la structure racine, et en déduire les contraintes qu'il est nécessaire de conserver dans le graphe

Nous ne disposons pas actuellement d'algorithme complet permettant d'effectuer ces différentes étapes. Certaines zones d'ombre restent à expliciter autour de l'identification des contraintes d'intervalles, nous présentons ici l'état de nos résultats.

Comme dans le cas précédent, nous supposerons que les partitions écrites sont *cohérentes* et que la *jouabilité* de la partition est assurée étant données les bornes des intervalles écrites.

9.4.4 Identification des contraintes d'intervalles

Principes

Rappelons que les contraintes d'intervalles sont dues soit à la définition d'une relation temporelle entre deux événements non-successifs, soit à des synchronisations entre événements de la partition. Les diverses origines des contraintes d'intervalles sont exposées dans la partie 7.3.5 du chapitre 7.

Pour présenter le mécanisme d'identification des contraintes d'intervalles, nous nous appuyons sur l'exemple de la figure 9.5.

Nous présentons sur la figure 9.6, le réseau de Petri résultant de la compilation de la structure de la figure 9.5. Ce réseau est obtenu de la même manière que pour le formalisme du "point d'orgue" en conservant les places considérées comme implicites.

Comme une contrainte d'intervalles est due à une synchronisation d'événements dans la partition, elle se signale dans le réseau de Petri par une transition disposant de plusieurs arcs entrants (nous appellerons ces transitions, des transitions de synchronisation), et elle se définit par une somme d'intervalles le long d'un chemin menant à cette transition de synchronisation. Nous le précisons dans la partie 7.3.5, une contrainte d'intervalles se définit par des égalités entre les lignes de temps les plus courtes menant à une synchronisation. Sur le réseau, il s'agit des chemins les plus courts d'une transition de synchronisation à une autre. Identifier les contraintes d'intervalles revient à identifier ces plus courts chemins dans le réseau de Petri.

La recherche de ces chemins s'appuie sur la structure de graphe sous-jacente au réseau de Petri. Pour simplifier la représentation, nous utiliserons donc des graphes d'événements. Les événements d'une partition forment les nœuds de ces graphes et un arcs relie deux sommets si l'intervalle séparant les deux événements est spécifié par une relation temporelle (implicite ou explicite). Comme les réseaux de Petri que nous manipulons sont des réseaux d'occurrences, on peut passer facilement d'un réseau au graphe d'événements en ne gardant que les transitions comme nœuds, et en remplaçant les séquences "arc-place-arc" séparant les transitions par des arcs étiquetés avec l'intervalle correspondant.

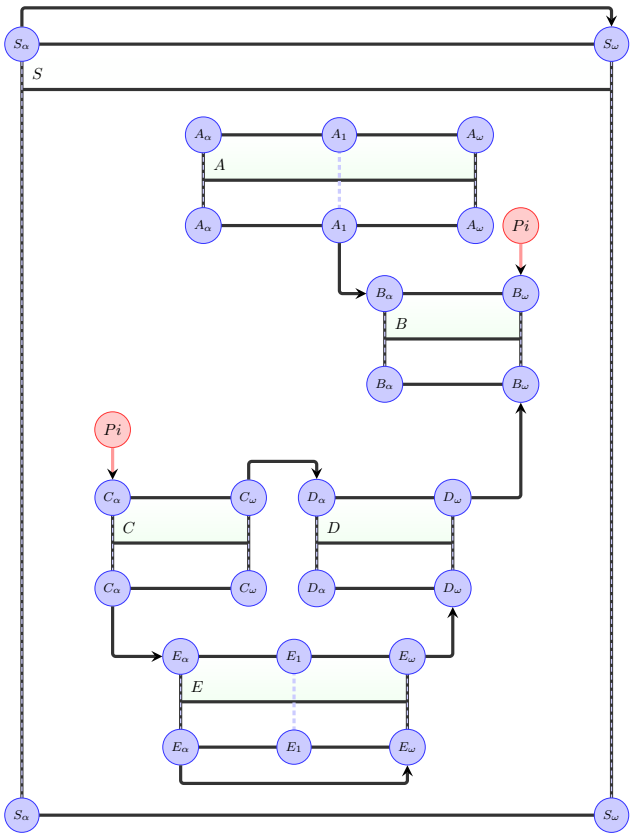


FIG. 9.5 – Un exemple de structure

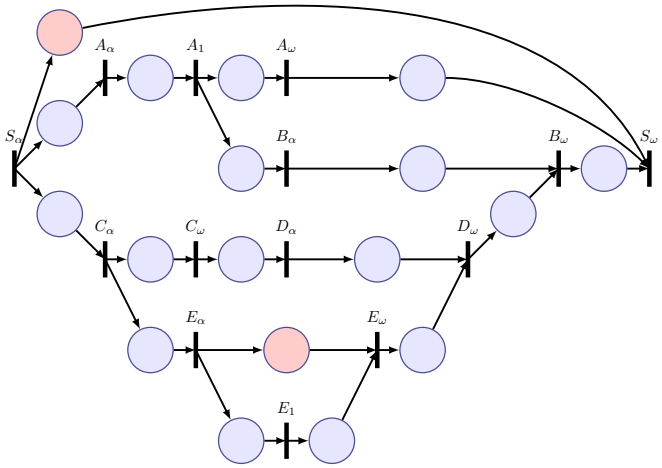


FIG. 9.6 – Le réseau de Petri issu de la compilation de la structure de la figure 9.5

Nous présentons sur la figure 9.7(a) le graphe d'événements associé à la structure de la figure 9.5.

L'identification des plus courts chemins menant d'une synchronisation à une autre peut être effectuée en cherchant à décomposer le graphe d'événements en composantes *série-parallèle* ([45]). En effet, dans le cas particulier où le graphe d'événements seraient lui-même *série-parallèle*, une composante parallèle représente l'égalité entre les sommes d'intervalles le long des chemins mis en parallèle. Et ainsi, une composante parallèle donne naissance à une contrainte d'intervalles pour chaque composante série mise en parallèle.

Naturellement, le graphe d'événements d'une structure n'est pas nécessairement *série-parallèle*. **La solution que nous adoptons est donc d'extraire du graphe d'événements un sous-graphe *série-parallèle*, dont on pourra déduire un ensemble de contraintes d'intervalles, auquel nous ajoutons ensuite les contraintes manquantes dues à la différence entre le graphe et le sous-graphe.** Le nombre de sous-graphes *série-parallèle* d'un graphe quelconque est potentiellement important. Dans notre cas, nous cherchons à construire un sous-graphe connexe conservant la source et le puits du graphe d'événement (début et fin de la partition), et qui soit maximal en nombre de nœuds.

Nous présentons sur les figures 9.7(b) et 9.7(c) les deux sous-graphes maximaux *série-parallèle* du graphe d'événements de la structure de la figure 9.5.

Une fois un sous-graphe *série-parallèle* maximal exhibé, la construction de l'ensemble de contraintes d'intervalles s'appuie sur la décomposition *série-parallèle* du sous-graphe. Sur la figure 9.8, nous nous appuyons sur le sous-graphe maximal de la figure 9.7(c) pour construire un ensemble de contraintes d'intervalles de base. Sur la figure 9.8(a), nous présentons l'arbre de décomposition *série-parallèle* du sous-graphe de la figure 9.7(c). Sur cet arbre, les feuilles sont les sommets du sous-graphe et les nœuds sont les opérateurs série (+) et parallèle (‖). Sur la figure 9.8(b), nous présentons sous forme de graphe de contraintes, l'ensemble des contraintes d'intervalles que l'on peut déduire de la décomposition *série-parallèle*.

Les contraintes d'intervalles sont les suivantes :

$$\begin{array}{lcl} C_1 & : & \Delta_s = \Delta_1 + \Delta_{A_1} + \Delta_{A_2} + \Delta_2 \\ C_2 & : & \Delta_s = \Delta_5 + \Delta_{10} + \Delta_7 + \Delta_4 \\ C_3 & : & \Delta_{10} = \Delta_C + \Delta_6 + \Delta_D \\ C_4 & : & \Delta_{10} = \Delta_8 + \Delta_E + \Delta_9 \\ C_5 & : & \Delta_E = \Delta_{E_1} + \Delta_{E_2} \end{array}$$

Comme annoncé précédemment, chaque composition parallèle est associée à une variable représentant l'intervalle entre ses événements extrêmes. Certains de ces intervalles sont déjà caractérisés dans la partition et donc présents dans le réseau de Petri (Δ_S, Δ_E). A l'inverse d'autres variables doivent être créées à la construction du graphe de contraintes (Δ_{10}).

Une fois construites ces contraintes issues de la décomposition du sous-graphe *série-parallèle*, il faut ajouter les contraintes issues des éléments du graphe d'événements non présents dans le sous-graphe *série-parallèle*.

Dans le cas de la structure de la figure 9.5, avec le sous-graphe *série-parallèle* de la figure 9.7(c), ces contraintes supplémentaires sont les suivantes :

$$\begin{array}{lcl} C_6 & : & \Delta_{11} = \Delta_{A_2} + \Delta_2 \\ C_7 & : & \Delta_{11} = \Delta_3 + \Delta_B + \Delta_4 \end{array}$$

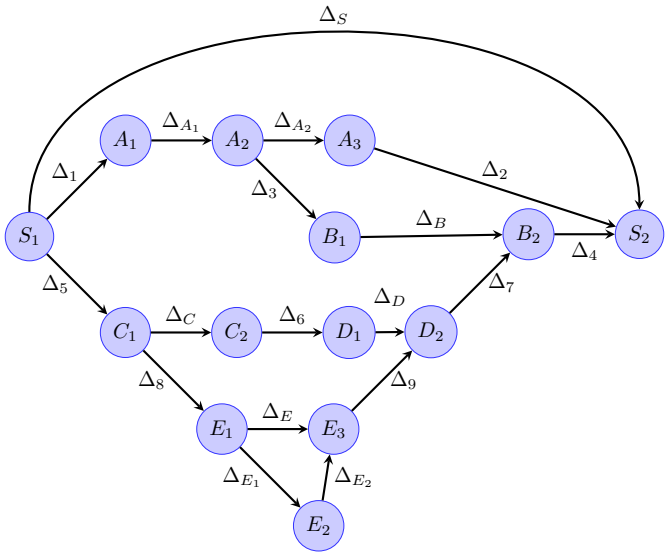
Comme lors de la décomposition du sous-graphe en composantes *série-parallèles*, on recherche des égalités entre des chemins. Une variable est créée pour représenter l'intervalle de temps séparant les événements extrêmes des chemins.

Le graphe de contraintes complet correspondant à la structure de la figure 9.5.

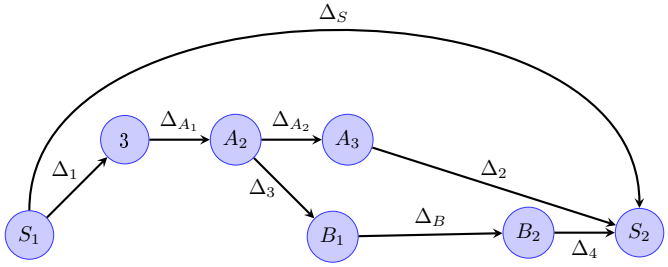
Algorithme de construction d'un sous-graphe *série-parallèle*

Le but de cet algorithme est de fournir un sous graphe *série-parallèle* maximal d'un graphe initial , c'est à dire disposant d'une source et d'un puits tel que tout nœud du graphe se situe sur un chemin depuis la source vers le puits. Nous donnons dans l'annexe B une version en "pseudo-code" de cet l'algorithme.

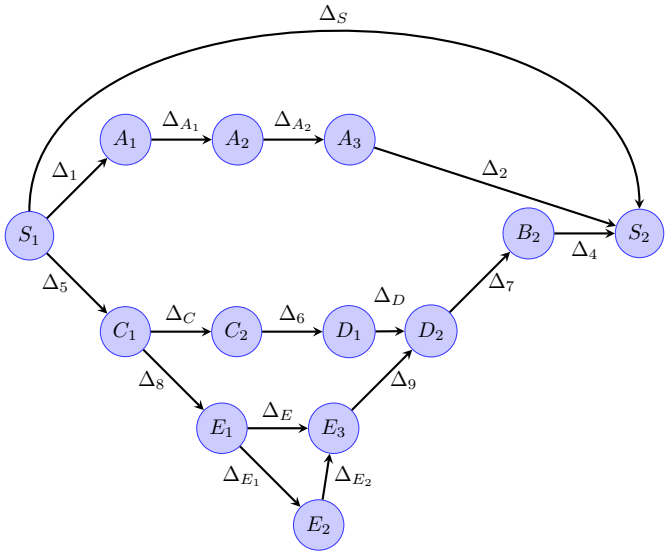
Nous présentons ici son idée directrice.



(a) Le graphe d'événements associé à la structure de la figure 9.5. Les point de contrôle sont représentés sur les sommets, un arc relie 2 sommets si l'intervalle entre les deux points est spécifié. L'arc étant étiqueté avec cet intervalle.

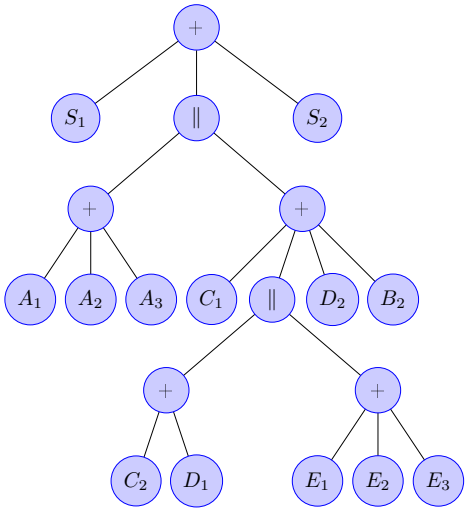


(b) Un sous-graphe *série-parallèle* maximal du graphe d'événements

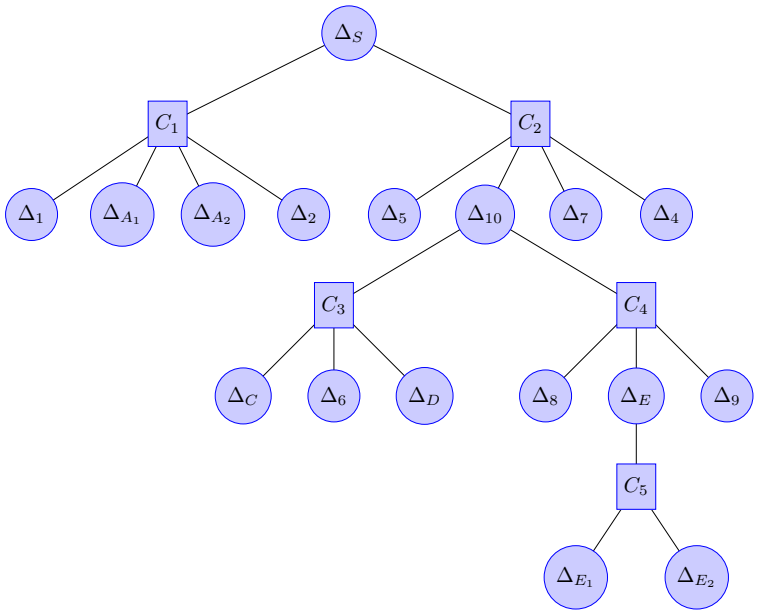


(c) Un autre sous-graphe *serie-parallèle* maximal du graphe d'événements

FIG. 9.7 – Le graphe d'événements de la structure de la figure 9.5 et ses sous-graphes maximaux *série-parallèle*



(a) L'arbre de décomposition *série-parallèle* du sous-graphe de la figure



(b) Le graphe de contraintes associé au sous-graphe de la figure

FIG. 9.8 – Arbre de décomposition *série-parallèle* du sous-graphe maximal de la figure 9.7(c) et graphe (arbre) de contraintes associé

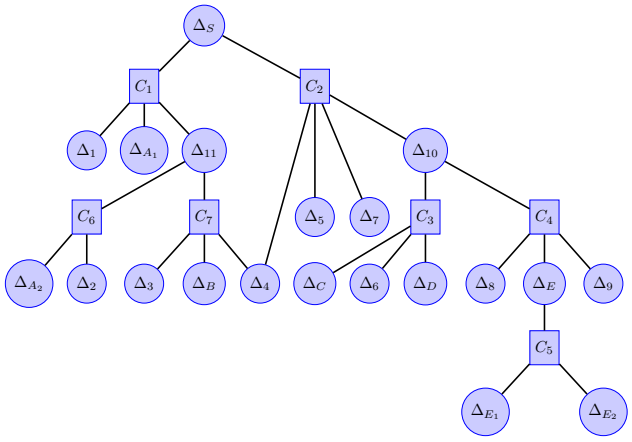


FIG. 9.9 – Le graphe de contraintes complet associé à la structure de la figure 9.5

Soient G un graphe, s et p sa source et son puits, on note C l'ensemble des chemins de s à p. Pour un chemin c de C, on note P_c l'ensemble des chemins de C partageant un préfixe de taille maximale avec c.

Définition 9.3 *Soit $c \in C$ $(c = c_1 \dots c_n)$ alors P_c se définit par :*

$$\begin{array}{l}\sigma = \sigma_1 \dots \sigma_m \in P_c \text{ si} \\ \exists i \in [1,n],\ t.q.\ c_1 \dots c_i = \sigma_1 \dots \sigma_i \\ et\ \nexists \alpha = \alpha_1 \dots \alpha_l\ t.q\ c_1 \dots c_{i+1} = \alpha_1 \dots \alpha_{i+1}\end{array}$$

De même, on note S_c l'ensemble des chemins de C partageant un suffixe de taille maximale avec c.
On cherche à construire un sous-ensemble E de C tel que :

$$\forall c \in E, P_c = S_c$$

et qui soit maximal en nombre de chemins.

Les nœuds et arcs formant les chemins de E constituent alors un sous-graphe *série-parallèle* maximal de G.

Pour la construction d'un tel ensemble, un algorithme "direct" construisant l'ensemble des chemins, avant de chercher à ajouter chaque chemin c au sous-graphe en comparant les ensembles P_c et E_c présente une complexité trop importante. Dans la version que nous proposons dans l'annexe B, la construction des chemins et le test de l'intégration dans le sous-graphe sont simultanés.

Une fois un sous-graphe série-parallèle maximal obtenu, la construction du graphe de contraintes correspondant à ce sous-graphe s'appuie sur un algorithme de décomposition de graphe série-parallèle.

Algorithme d'identification des contraintes hors sous-graphe *série-parallèle*

Une fois un sous-graphe *série-parallèle* maximal obtenu, la prise en compte des contraintes liées aux éléments non présents dans le sous-graphe consiste à trouver directement des égalités entre plus courts chemins impliquant des éléments non présents dans le sous-graphe *série-parallèle*.

Soient G un graphe, G_1 un sous-graphe maximal *série-parallèle* de G et E l'ensemble des nœuds et arcs de G n'appartenant pas à G. Pour un nœud n de G dont l'un des arcs sortant appartient à E, on cherche à caractériser le plus court chemin c tel que :

$$c = c_1 c_2 \dots c_k \text{ t.q. } c_1 = n \text{ et } c_k \in G \text{ et } \forall i \in [2, k-1] c_i \in E$$

Un fois ce plus court chemin obtenu, on cherche le plus court chemin de n à c_k dans G, i.e. :

$$c' = c'_1 c' 2 \dots c'_{k'} \text{ t.q. } c'_1 = n \text{ et } c'_{k'} = c_k \text{ et } \forall i [1, k'], c'_i \in G$$

Ces recherches de chemins se font sans prendre en compte l'orientation des arcs. Pour reprendre l'exemple de la figure 9.7, en considérant que le sous-graphe *série-parallèle* construit est celui de la figure 9.7(c), la recherche d'un chemin dans l'ensemble E, va construire le chemin :

$$A_2B_1B_2$$

Une fois B_2 connu, le chemin dans G qui est construit est :

$$A_2A_3S_2B_2$$

Lorsque ces deux chemins sont déterminés, on peut s'appuyer sur l'orientation des arcs pour trouver l'égalité des sommes d'intervalles sur les 2 chemins :

$$A_2B_1B_2S_2 \text{ et } A_2A_3S_2$$

Cette égalité entre ces chemins permet la création d'une contrainte par chemin, en faisant intervenir une variable représentant l'intervalle séparant les événements situés aux extrémités des chemins (n et c_k). Les nœuds et arcs de E formant le chemin c, sont alors retirés de E et ajoutés à G. L'opération est alors réitérée jusqu'à ce que E soit vide.

Potentiellement, deux nœuds de G peuvent être connectés par plusieurs chemins d'éléments de E, qui seront considérés comme plus court aux différentes étapes de la construction du graphe de contraintes. Dans ces conditions, la variable créée pour représenter l'intervalle entre les deux nœuds va être partagée par plus de 2 contraintes d'intervalles.

Nous ne fournissons pas de version des algorithmes permettant la résolution de ces étapes, cependant on peut s'appuyer sur la description que nous venons d'en faire pour implémenter ces étapes à l'aide d'un parcours en profondeur pour trouver le plus court chemin dans E, et un algorithme de plus court chemin pour trouver celui dans G (comme par exemple le classique algorithme de Dijkstra [41]).

9.4.5 Algorithme de propagation

L'élaboration d'un algorithme de résolution s'appuyant sur le graphe de contraintes part de plusieurs constats :

- les valeurs des intervalles avant l'exécution forment une solution du problème
- le déclenchement d'un point d'interaction constitue une perturbation de la solution initiale
- les méthodes de propagation sont à sorties multiples
- le graphe de contraintes peut présenter des supports de cycles
- le choix d'une méthode de propagation au niveau d'une contrainte c est déterminé par la variable perturbée déclenchant la propagation à travers c, ce qui implique des changements de sens de propagation selon le point d'interaction déclenché
- l'ajout ou la suppression d'une contrainte ne peut pas intervenir pendant l'utilisation de l'algorithme
- l'ajout ou la suppression d'un point d'interaction est impossible pendant l'utilisation de l'algorithme

La présence de cycles dans le graphe oriente vers l'utilisation d'algorithmes de propagation capables de gérer ce genre de problème. Si ceux-ci existent (*SkyBlue*, *UltraViolet*...), leur mise en œuvre est assez lourde. En outre les remarques ci-dessus nous indiquent que ce type d'algorithme dépasse largement le problème posé. D'une part l'absence de choix pour une méthode de propagation au niveau d'une contrainte rend caduc l'utilisation d'un algorithme de planification de méthodes pour éviter les phénomènes de cycles et de conflits.

L'orientation du graphe de contraintes pour la propagation d'une perturbation due à un point d'interaction se fait directement comme en témoigne les exemple de la figure 9.10. Nous reprenons ici l'exemple de la figure 9.5, le graphe de contraintes obtenu après analyse du réseau de Petri est orienté selon les propagations dues aux points d'interaction. Dans le cas de la figure 9.10(a), la perturbation provient de la modification de Δ_5 par le déclenchement du point d'interaction sur $start(C)$. Comme Δ_5 fait partie du membre droit de C_2, la propagation se fait vers le membre gauche de C_2 et les autres variables de son membre droit (Δ_4 et Δ_7). L'orientation du reste du graphe se fait simplement selon cette règle au niveau des autres contraintes. On peut observer l'apparition d'un conflit sur Δ_4 due au support de cycle présent

dans le graphe de contraintes. On peut remarquer que ce conflit aurait pu porter sur une autre variable en changeant l'ordre dans lequel nous avons considéré les contraintes lors de l'orientation.

Intuitivment, comme les contraintes que nous utilisons sont linéaires, la variable sur laquelle porte le conflit ne semble guère avoir d'importance. En particulier, cela ne semble pas avoir d'importance sur l'apparition d'une incompatibilité à cause du conflit.

La figure 9.10(b) qui oriente le graphe en fonction de la perturbation due au point d'interaction sur $end(B)$ procède de la même manière mais fait apparaître un phénomène important : la chronologie entre les variables. En effet, on peut remarquer que la perturbation sur Δ_B ne se propage pas vers Δ_8 car cette variable sera écoulée lorsque Δ_B sera modifiée. Par contre, l'indéterminisme sur les ordre des événements des différentes lignes de temps implique d'envisager des propagations "dans le doute". Ainsi la perturbation de Δ_4 peut conduire à celle de Δ_{10} car rien ne permet d'affirmer que l'événement $start(C)$ (début de Δ_{10}) se sera produit au moment de la modification de Δ_B. Au moment de l'exécution il est probable que $start(C)$ se soit effectivement produit au moment du déclenchement de $end(B)$, et dans ce cas la perturbation de Δ_4 ne pourra se propager vers Δ_{10}, mais c'est seulement à ce moment que nous en serons assurés. On peut remarquer que la propagation ne se fait pas vers Δ_5. En effet, la valeur de celui-ci étant directement déterminé par le déclenchement d'un point d'interaction, soit il sera déjà écoulé lors du déclenchement de $end(B)$, soit le déclenchement interactif de $start(C)$ viendra modifier sa valeur par la suite.

Plusieurs remarques par rapport à ces orientations.

Une orientation basée sur une analyse plus fine nous permettrait d'écarter plus de propagations que ce nous envisageons dans ces deux exemple. Par exemple dans la figure 9.10(b), les variables Δ_1 et Δ_{A_1} ne pourront pas être modifiées car ces intervalles seront écoulés lors du déclenchement de $end(B)$. Dans notre cas, comme l'orientation au niveau de C_1 se fait en considérant la perturbation de Δ_S, on ne peut exclure la propagation vers ces variables.

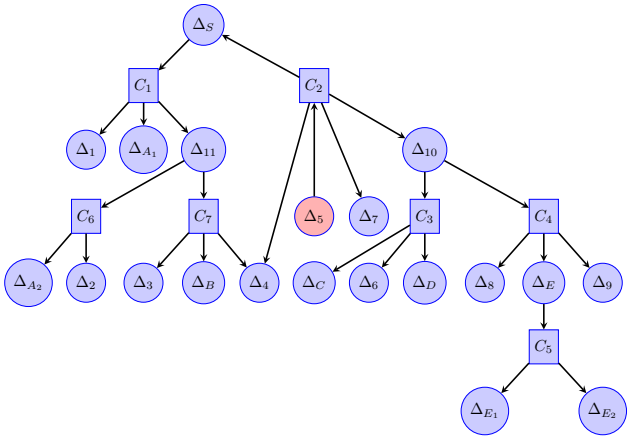
De plus, il nous faut mentionner une autre possibilité de conflit qui se produit lorsque deux contraintes partagent 2 variables ou plus. La propagation au niveau d'une contrainte peut conduire à la modification de plusieurs variables partagées et donc à la perturbation simultanée de plusieurs variables de la seconde contrainte. Comme les méthodes de propagation envisagées ne prennent qu'une seule variable en entrée, l'orientation du graphe selon la perturbation de l'une des variables partagées fera apparaître des conflits au niveau des autres. Cette situation se produit également lorsqu'un événement dynamique modifie plusieurs intervalles.

Outre l'orientation déterminée du graphe de contraintes pour un point d'interaction, les constats qu'aucune contrainte ou qu'un aucun point d'interaction ne pourra être ajouté ou supprimé oriente définitivement notre choix d'algorithme vers un algorithme de compilation de problème de contraintes tel "Projection Based Compilation" ([18], [51]). Cet algorithme cherche à produire un code efficace pour la résolution d'un problème de contraintes donné. La motivation de cet algorithme est la gestion d'affichage de fenêtres graphiques dont la disposition est spécifiée au travers de contraintes. Les relations entre les différentes fenêtres sont données une fois pour toutes, mais les actions d'un utilisateur peuvent modifier un des paramètres (taille d'une fenêtre par exemple) nécessitant alors de résoudre à chaque fois le problème de contraintes. L'idée est donc d'analyser le problème de contrainte afin de produire du code (Java en l'occurrence, des applications web étant visées) qui résolvent le problème donné sans faire appel à un moteur de contraintes. Cette compilation repose sur le fait que le problème de contraintes restera inchangé au cours de l'utilisation et que les interactions possibles sont connues. Un des intérêts majeurs de cette solution étant une certaine maîtrise des performances en termes de temps de réponse.

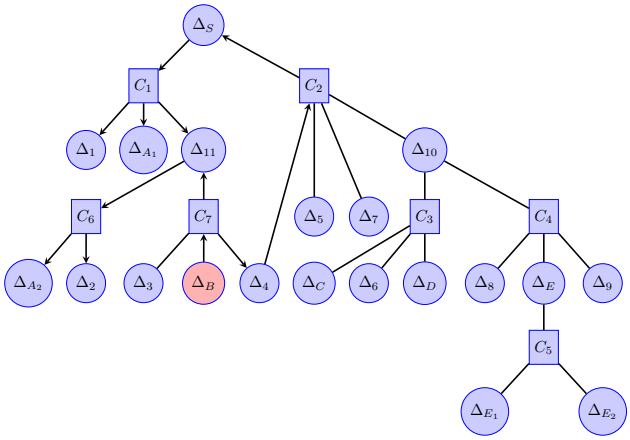
On le voit, les questions abordées par cet algorithme sont proches de celles posées par le notre. Cependant son application directe dans notre cas n'est pas possible car les problèmes de contraintes qu'il résout sont différents des nôtres (hiérarchie entre les contraintes, absence de méthodes de propagation ...)

Bien que nous ne disposons pas d'algorithme complet pour la compilation des propagations de perturbation, nous proposons néanmoins le principe suivant :

- orienter le graphe de contraintes pour chaque point d'interaction
- déduire pour chaque point d'interaction une suite d'opérations permettant de réagir au déclenchement du point d'interaction



(a) L'orientation du graphe de contraintes pour la propagation de la perturbation due au point d'interaction sur $start(C)$



(b) L'orientation du graphe de contraintes pour la propagation de la perturbation due au point d'interaction sur $end(B)$

FIG. 9.10 – Les orientations du graphe de contraintes associé à la structure de la figure pour chacun de ses points d'interaction

Pendant l'exécution, lorsque qu'un point d'interaction est déclenché, les opérations nécessaires à la propagation sont lancées. Les conflits sont simplement vérifiés, si une variable que l'on envisageait de modifier ne peut plus l'être, la propagation n'est pas effectuée à travers elle.

9.4.6 Jouablité

Comme nous l'avons précisé précédemment, la jouabilité des partitions pose problème lorsque la rigidité des intervalles est acceptée. Cependant, les choix de dates des points d'interaction peuvent conduire à invalider la jouabilité d'une partition. La jouabilité d'une partition est directement liée aux domaines des variables d'intervalles. Concrètement, soient Δ_1 et Δ_2 de domaines respectifs $[\Delta_{1_{min}}, \Delta_{1_{max}}]$ et $[\Delta_{2_{min}}, \Delta_{2_{max}}]$, le choix d'une valeur pour Δ_1 dans $[\Delta_{1_{min}}, \Delta_{1_{max}}]$ peut conduire à ce que le choix de certaines valeurs de $[\Delta_{2_{min}}, \Delta_{2_{max}}]$ ne conduisent plus à une solution du système de contraintes d'intervalles.

Ainsi, pour maintenir la jouabilité d'une partition au cours de son exécution, il convient de réduire les domaines des variables non encore déterminée après le déclenchement de chaque point d'interaction. Là encore, la question de l'algorithme à utiliser pour effectuer cette tache se pose. L'utilisation d'un algorithme de réduction de domaine classique tel *Indigo* se heurte aux éventuelles cycles dans le graphe de contraintes nous renvoyant à l'utilisation de solutions lourdes en terme de temps de calcul.

Pour remédier à ce problème une solution pourrait être de s'appuyer sur la convexité des domaines de variables pour contraintes linéaires.

En effet, soit une contrainte linéaire portant sur des variables à valeurs réelles définie par :

$$C : \sum_{i=1}^{n} a_i.\Delta_i = b$$

Soient :

$$(\delta_1 \dots \delta_n) \in \mathbb{R}^n$$

$$(\delta'_1 \dots \delta'_n) \in \mathbb{R}^n$$

deux solutions de C.

Alors :

$$\begin{aligned} &\forall \alpha \in [0,1], \\ &\sum_{i=1}^{n} a_i.(\delta_i + \alpha.(\delta'_i - \delta_i)) \quad = \sum_{i=1}^{n} a_i.\delta_i + \alpha.(\sum_{i}^{n} a_i\delta'_i - \sum_{i}^{n} a_i\delta_i) \\ &\qquad\qquad\qquad\qquad\qquad\qquad = b \end{aligned}$$

Par conséquent, une solution pourrait consister à trouver après chaque déclenchement d'un point d'interaction, deux solutions extrêmes pour les variables dont la valeur n'a pas encore été fixée.

On peut reconnaître à ces solutions une certaine lourdeur. A défaut de solution réaliste, le maintien de la jouabilité peut ne pas être assuré. Dans ce cas, les domaines des variables garderaient leurs valeurs initiales. Il serait alors de la responsabilité de l'interprète de rester dans les limites d'une solution acceptable au regard des contraintes d'intervalles.

9.4.7 La machine ECO

Nous présentons brièvement sur la figure 9.11 la modification de la machine ECO pour accueillir un algorithme de résolution de contraintes d'intervalles au cours du déroulement de la partition. Celui prend naturellement place au niveau de l'ordonnanceur de la machine et interagit avec le réseau de Petri. Lorsqu'un point d'interaction est déclenché, le franchissement de la transition interactive correspondante déclenche une résolution du système de contraintes. Cette résolution est bien sûr celle associée au point d'interaction qui vient d'être déclenché. En retour, une fois les valeurs d'intervalles calculés, celles-ci sont fournies en retour au réseau de Petri.

Le fonctionnement d'une telle machine soulève quelques questions par rapport au temps de calcul nécessaire à la résolution du système de contraintes. Notamment, si la résolution est trop longue, il n'est

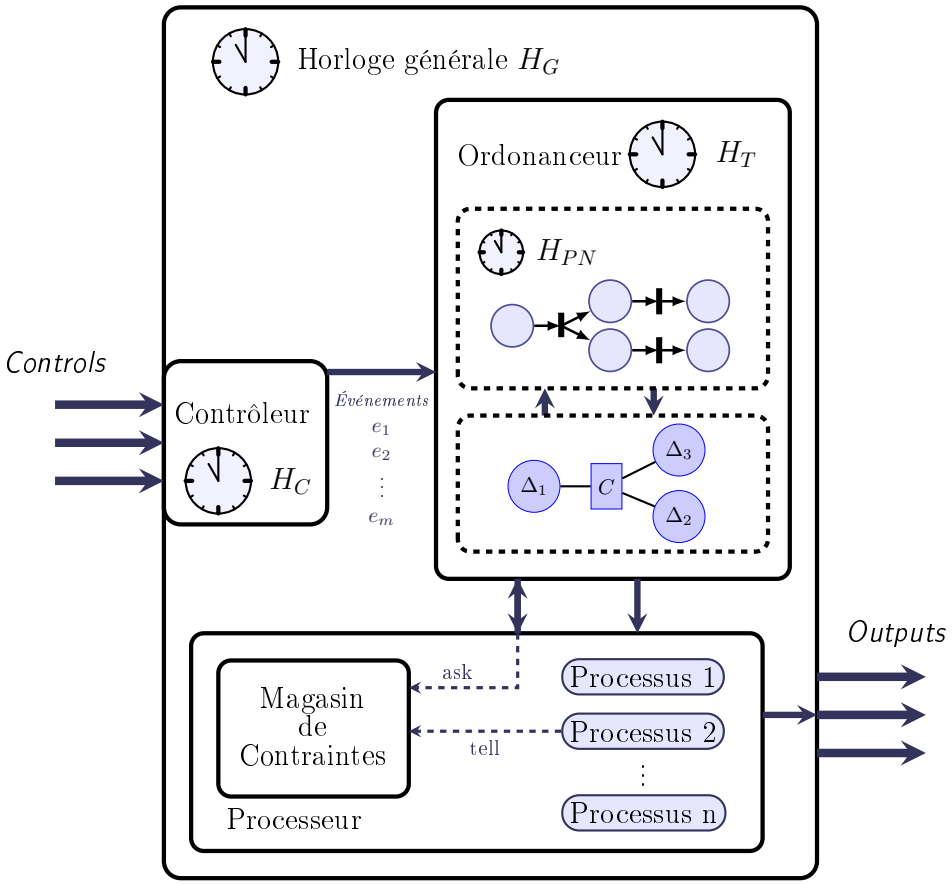


FIG. 9.11 – La machine *ECO* contenant un graphe de contraintes. La propagation des perturbations dans le graphe est pilotée par le franchissement des transitions dans le réseau de Petri. En retour, les valeurs d'intervalles nouvellement calculées sont communiquées au réseau de Petri.

pas impossible que des valeurs d'intervalles que celle-ci vient de produire ne soient plus applicables au réseau, si ces intervalles ont commencé à s'écouler pendant la résolution. Un système d'attente d'une solution sera peut-être nécessaire.

9.5 Formalisme complet

En conclusion de ce chapitre, nous proposons quelques pistes à explorer pour modéliser les partitions du formalisme complet et en déduire une machine *ECO* correspondante.

9.5.1 Contraintes

Dans la version du formalisme que nous venons d'étudier, la possibilité de définition de contraintes est limitée aux contraintes d'intervalles. Or dans le formalisme complet des partitions interactives, d'autres variables et d'autres types de contraintes sont envisagés. Pour intégrer ceux-ci au modèle d'exécution, la construction du graphe de contraintes que nous avons proposée ci-dessus pourrait être complétée par l'ajout de ces variables (quanta temporels, entrée/sortie des processus) et des contraintes les impliquant. Les contraintes proposées pour ces variables disposant de méthodes de propagation bien établies, l'orientation du graphe de contraintes pourrait tout à fait se dérouler comme indiquer précédemment.

9.5.2 Hiérarchie

La première solution que nous ayons envisagée concernant la hiérarchie fut de l'abolir au niveau de la représentation sous forme de réseaux de Petri. Il s'agissait donc de représenter chaque structure par un sous-réseau, et d'intégrer tous ces sous-réseaux dans un réseau général. Cette solution était motivée par la présence des relations trans-hiérarchiques. Avec cette approche, celles-ci pouvaient être traitées comme des relations temporelles internes à une structure au moment de la compilation. Cependant, cette "mise à plat" pose deux problèmes majeurs :

- l'impossibilité de manipuler le quantum temporel d'une structure en particulier. En effet, si toutes les structures se voient mêler dans un seul niveau hiérarchique du réseau de Petri, seul le cadencement général se trouve alors modifiable. Sa modification induit alors une modification générale.
- l'interruption d'une structure particulière est impossible, sauf à mettre en place des mécanismes très complexes pour suivre l'évolution des transitions du réseaux représentant les événements de chaque structure.

Une solution plus réaliste semble donc d'utiliser une modélisation hiérarchique s'appuyant sur les réseaux de Petri hiérarchisés ([76]). Dans cette solution, chaque structure est transformée en un sous-réseau par compilation. Au niveau d'un réseau représentant une structure, ses éventuelles structures enfant sont représentées par des places complexes. La construction du réseau hiérarchique global devra donc s'effectuer de manière ascendante depuis les structures de plus bas niveau dans la partition.

La possibilité d'interrompre une structure se modélise alors simplement par le mécanisme présentée dans la section 3.5.3.

Si toutes les horloges des sous-réseaux sont synchrones avec l'horloge du réseau de plus haut niveau (H_{PN}), il est possible de modifier le pas de l'une d'entre elles pour modéliser le changement de quantum temporel d'une structure particulière.

Relations trans-hiérarchiques

La modélisation des relations trans-hiérarchiques dans le cadre des réseaux de Petri hiérarchisés nécessite que l'on s'y arrête un instant. Chaque sous-réseau étant considéré comme autonome, il n'est pas possible de construire des arcs entre des éléments de deux sous-réseaux différents. Il faut mettre en place des mécanismes de messages entre les sous réseaux.

Nous proposons sur la figure 9.12 un exemple de ce mécanisme pour la synchronisation de deux transitions appartenant à des sous-réseaux différents. Sur cet exemple, les transitions t_1 et t_2 doivent être tirées simultanément. Pour permettre cette synchronisation, il faut que le franchissement de chacune d'entre elles soit conditionné par la possibilité de franchir l'autre. Le mise en place d'un tel conditionnement

directement les t_1 et t_2 conduirait une situation de blocage mutuel. C'est pourquoi il nous faut dupliquer t_1 et t_2 en t'_1 et t'_2 dont le but est d'informer l'autre sous-réseau de la possibilité de franchir la transition qu'ils dupliquent.

Les arcs qui précèdent t'_1 et t'_2 ont les mêmes caractéristiques que ceux qui précèdent t_1 et t_2. La présence d'un jeton dans les places p_1 et p_2 conduit à la production d'un jeton dans les place p'_1 et p'_2 au travers d'un message envoyé à l'autre sous-réseau.

Il est important que les arcs précédant t'_1 (resp. t'_2) soient placés dans le même sous-réseau que t_1 (resp. t_2), car leur compteur associé se doit d'évoluer à vitesse de l'horloge du sous-réseau auquel la transition qu'ils dupliquent appartient. On peut remarquer les réseaux ainsi créés restent des réseaux d'occurrences.

Graphe de contraintes

L'utilisation de sous-réseaux de Petri suggère l'utilisation de sous-réseaux de contraintes. L'idéal serait en effet de pouvoir associer à chaque sous-réseau de Petri, un sous-graphe de contraintes associé. Cependant, les relations temporelles trans-hiérarchiques et certaines contraintes (contraintes entre les quanta temporels de différentes structures) impliquent des liens entre les graphes de contraintes associés aux sous-réseaux. Rappelons que les variables d'une sous-structures apparaissent dans l'ensemble des variables de sa structure parent. L'approche que nous nous proposons d'adopter à ce propos est de s'appuyer sur la construction ascendante du réseau de Petri hiérarchisé, pour à chaque niveau, construire un sous graphe de contraintes pour le niveau considéré, et analyser ce dernier pour déterminer qu'elles variables se doivent d'être présentes au niveau supérieur. L'objectif est d'isoler au maximum les sous-graphes de contraintes pour limiter la propagation d'une perturbation et ainsi le temps de calcul.

9.5.3 Structures logiques

L'utilisation de réseaux hiérarchisés et la représentation de chaque structure par un réseau propre, permet la modélisation des structures logiques et leur intégration dans la hiérarchie.

En effet, la mdélisation des structures logiques par des automates d'une part, et les liens entre réseaux de Petri et automates d'autre part, autorisent la modélisation des structures logiques par les réseaux de Petri.

La figure 9.13 présente la modélisation d'une structure logique par un réseau. Nous reprenons l'exemple de structure logique proposé dans la section 5.5.1. Sa modélisation en réseau de Petri se fait de la manière suivante : chaque enfant est représenté par deux transitions, une pour son début et une pour sa fin (rappelons qu'ils ne peuvent avoir que ces deux points de contrôle). Ces deux transitions sont séparées par une place représentant la phase de déroulement de l'objet enfant. De plus, une place précédant la transition de début et une autre succédant la transition de fin de l'enfant sont créées (p_1, p'_1 pour S_2, p_2 et p'_2 pour T_1). L'arc séparant p_1 et $start(S_2)$ est associé avec un intervalle $[0, \infty]$ permettant ainsi un franchissement immédiat de $start(S_2)$ dès la production d'un jeton dans p_1. Ces places précédant et succédant les représentations des objets enfants permettent de modéliser les relations logiques entre ceux-ci. Ainsi, p'_1 permet d'attendre le choix entre la redirection vers $start(S_2)$ et $start(T_1)$. Le franchissement de la transition t_1 est ainsi conditionné par la véracité de la formule a, tandis que celui de t_2 l'est par la véracité de la formule b. Le choix par défaut au niveau de la relation logique sortant de S_2 conduit à débuter T_1 (ceci pour permettre de sortir de la structure en suivant une succession de choix par défaut). Ainsi l'arc qui même de p'_1 à t_2 est-il associé avec l'intervalle $[\Delta_{min}, \Delta_{max}]$, tandis que celui menant de p'_1 à t_1 est associé à l'intervalle $[\Delta_{min}, \infty]$. Par conséquent ni t_1, ni t_2 ne pourra est franchie avant l'écoulement de Δ_{mi}, en revanche c'est t_2 qui sera franchie si la durée Δ_{max} s'est écoulée sans que rien de se soit produit.

Enfin, une place d'entrée et de sortie de la structure sont créées.

On peut remarquer que le réseau n'est plus un réseau d'occurrences.

La compilation d'une partition se fait alors de manière ascendante mais en distinguant le type de structure à chaque niveau, et donc le type de compilation.

Dans cette construction, un réseau modélisant une structure linéaire est toujours un réseau d'occurrences et ce même si elle dispose de structures logiques parmi ses enfants.

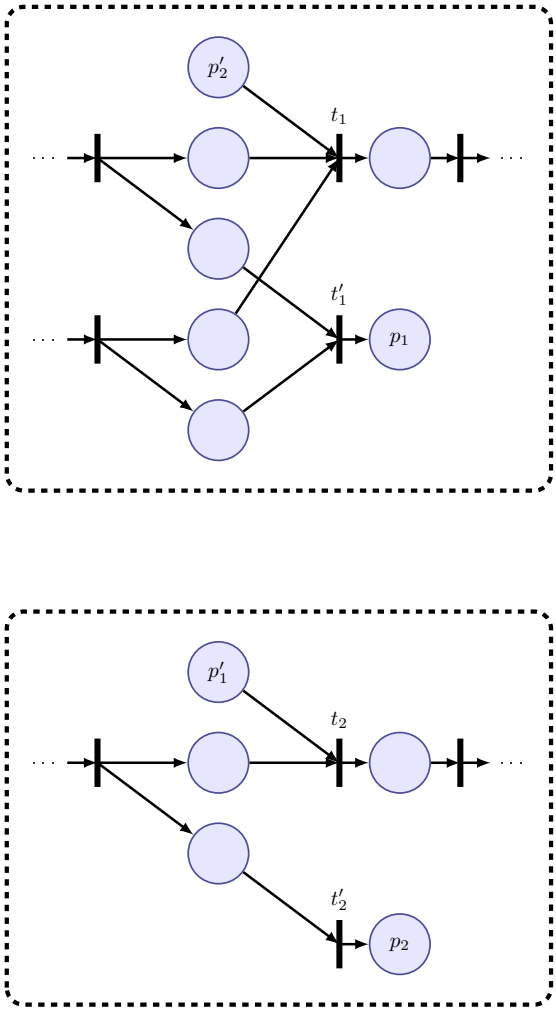
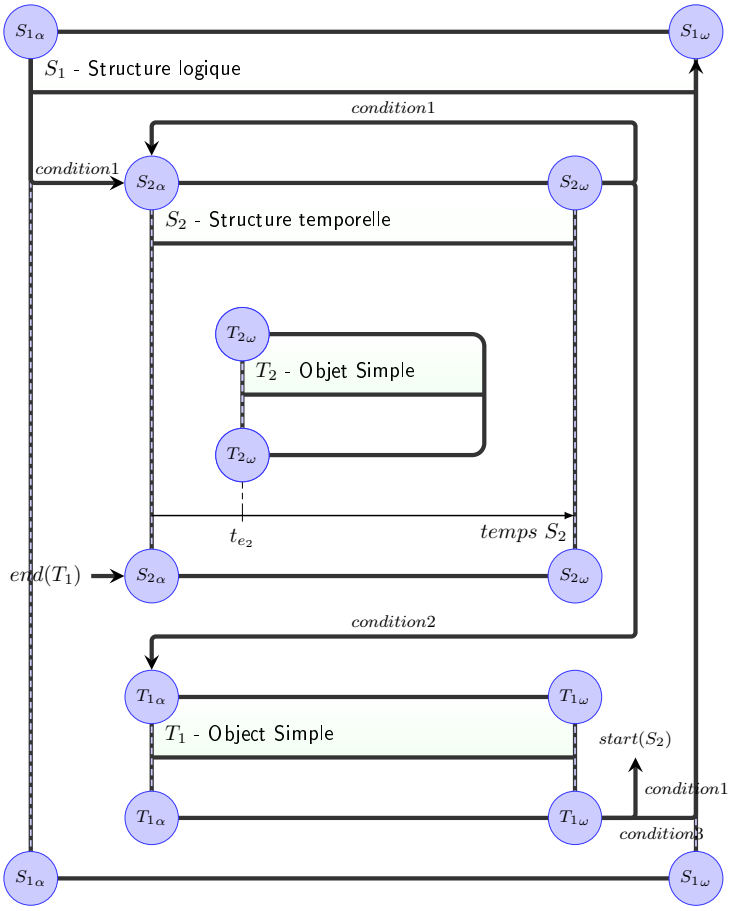
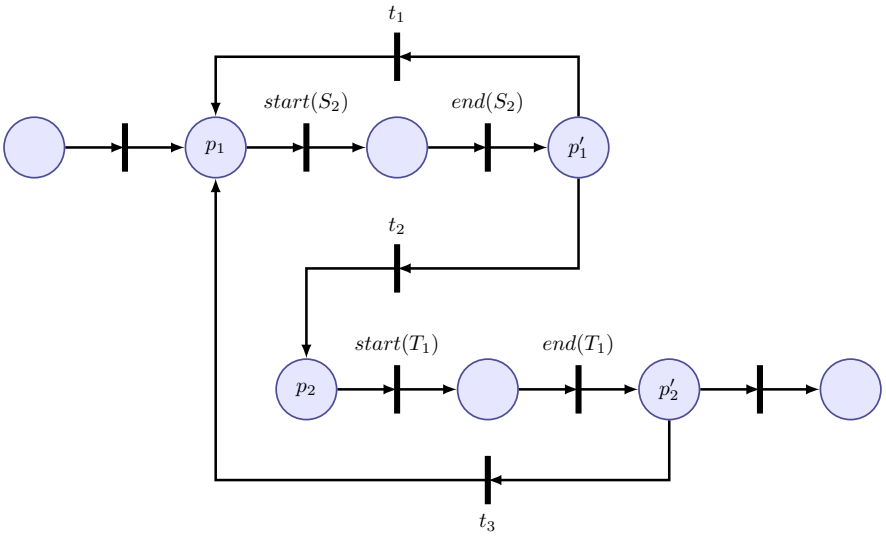


FIG. 9.12 – Un exemple de synchronisation entre deux transitions appartenant à deux sous-réseaux différents. Chacune des transitions est ici représentée dand le contexte de son propre sous-réseau. L'objectif est de synchroniser les transitions t_1 et t_2. Ces deux transitions sont donc dupliquées (t'_1 et t'_2) ainsi que les arcs et places qui les précèdent. La transition t'_1 est franchissable en même temps que t_1, de même entre t'_2 et t_2. Mais t'_1 est franchie dès que t_1 est franchissable, ce qui conduit à la production d'un jeton dans p_1. Cette production de jeton s'accompagne d'une production, par échange de message, d'un jeton dans p'_1. Le franchissement de t_2 étant conditionné à la présence d'un jeton dans p'_1, t_2 n'est franchissable que si t_1 l'est relativement à ses prédécesseurs dans son propre sous-réseau. L'arc séparant p'_1 et t_2 est associé à un intervalle de tir $[0, \infty]$. Symétriquement, t_1 n'est franchissable que si t_2 l'est relativement a ses prédécesseurs dans son propre sous-réseau. Au final, les transitions sont franchissables (et franchies) simultanément.



(a) Un exemple de structure logique



(b) La modélisation de la structure logique par un réseau de Petri. Chaque enfant de la structure est représenté par une transition de début et une transition de fin séparées par une place, simple pour un objet simple, complexe pour une structure. En outre, une place précédant et une place succédant l'enfant sont créées. Ces places permettent de modéliser les relations logiques entre les différents enfants de la structure. Après la transition $end(S_2)$, une place permet d'attendre le choix entre le franchissement de la transition pour revenir à $start(S_2)$ ou d'enchaîner avec $start(T_1)$. De plus une place de début et de fin de la structure sont créées.

FIG. 9.13 – La modélisation d'une structure logique par un réseau de Petri

9.6 Bilan

Nous avons exposé une solution complète pour la modélisation des partitions dans un formalisme restreint possédant les caractéristiques suivantes :

- hiérarchie limité à un niveau la structure étant linéaire
- la structure racine est définie avec le comportement "point d'orgue"
- tous les intervalles sont souples
- la contrainte globale limitant le nombre d'objets simultanément exécutables est autorisée
- les processus associés aux objets disposent de valeurs de calcul avant l'exécution de la partition, il n'est pas possible de faire des branchements entre ceux-ci

Cette modélisation, basée sur les réseaux de Petri, nous a permis de définir une implémentation de la machine *ECO* pour ce formalisme.

Nous avons ensuite présenté la trame générale d'une solution pour un formalisme dans le lequel les intervalles ne sont plus souples, le comportement de la structure racine pouvant être quelconque.

Choisissant une solution utilisant des réseaux de Petri additionés d'un système de propoagation de contraintes par perturbation, nous avons donné une manière d'identifier les contraintes d'intervalles en utilisant la décomposition *série-parallèle* appliquée au réseau de Petri modélisant la partition. Nous avons en ensuite déduit une construction du graphe de contraintes. Concernant l'algorithme de résolution, optant pour une compilation préalable de la propagation dans le graphe de contraintes, nous avons proposé une analyse des points d'interactions de la partition permettant d'orienter le graphe de contraintes pour chacun d'entre eux, et d'en tirer les opérations nécessaires à la pertubation associée à chauqe point d'interaction.

Enfin, nous avons proposé des pistes pour modéliser la hiérarchie et les relations trans-hiérarchiques dans les partitions, ainsi que les structures logiques.

Quatrième partie

Applications et Perspectives

Résumé

L'idée initiale qui a donné naissance au formalisme des partitions interactives, consistait à développer un outil pour les compositeurs de musiques électro-acoustique "écrite", pour permettre l'interprétation de leurs œuvres par des musiciens. Naturellement, cette volonté originelle donne lieu à la principale application de ce travail : un outil de composition assistée par ordinateur, également capable d'exécuter interactivement les partitions créées à travers lui. Un tel outil a été implémenté dans l'environnement des *Maquettes* du logiciel *OpenMusic*. Cependant la possibilité de représenter différents types de musique au travers du formalisme de partitions interactives nous permet d'envisager d'autres applications autour de la musique instrumentale notamment dans le domaine de la pédagogie. Une adaptation du formalisme à la pratique théatrâle est également expérimentée.

Ces appliquations s'appuient sur un format XML correspondant au formalisme, qui permet d'encoder les partitions et de faire l'interface avec d'autres formats musicaux (*MusicXML*). Ce format pourrait servir de base à un éventuel format général d'encodage de partition musicale.

Nous détaillons dans un premier temps les applications du formalisme et des outils auxquels il a donné naissance avant, puis nous proposons quelques perspectives pour la suite de ce travail.

Abstract

At the begining of this work was the aim to develop a tool for the composers of electro-acoustic music, that could allow the interpretation of their pieces. Therefore the main application of the formalism is the development of a tool for computer assisted composition, which can execute the pieces that are written with it. Such a tool has been implemented into the *Maquettes* environment of the *OpenMusic* software. But the capacity of the formaism to represent other types of music, allows us to imagine other applications. Mostly in the context of the instrumental music, these applications deals with pedagogical uses. An adjustment of the the formalism to the context of the living art is in progress.

All the applications are based on an XML format wich is a translation of the formalism. It allow the encoding of the interactive scores, but it can also be used as an interface with other musical formats (*MusciXML*). This format could be the base of a general encoding of music.

First, we present the applications of the formalism and then we propose some axis for the future of this work

Chapitre 10

Applications

Nous exposons dans ce chapitre les applications de notre formalisme, en commençant par présenter l'implémentation d'un outil d'édition et d'exécution dans le logiciel *OpenMusic*.

La mise en pratique des applications envisagées se fondent sur cette implémentation, depuis la création de pièces interactives jusqu'à l'importation et l'exécution de partitions instrumentales classiques.

De plus, nous évoquons le développement du projet *Virage* visant à adapter notre formalisme à la création et l'exécution de conduites interactives pour la régie numérique de spectacles vivants.

10.1 Musique

Naturellement les applications premières du formalisme sont en direction de la musique, et en particulier la composition assistée de pièces interactives. Nous avons donc engagé l'implémentation du modèle pour fournir un outil aux compositeurs.

10.1.1 Implémentation dans OpenMusic

Les premiers développements avaient pour but d'étendre les capacités du logiciel *Boxes* pour y permettre la création et l'exécution de partitions interactives. La volonté d'ouvrir notre formalisme et d'intégrer un environnement de composition plus vaste, nous ont conduits à nous tourner vers *OpenMusic* et ses *Maquettes*. Les *Maquettes* proposent un environnement de composition assez proche de celui de *Boxes*, dans lequel les objets sont représentées sous forme de "boîtes" que l'on peut venir positionner sur une ligne de temps représentée par l'axe des abcisses. L'axe vertical n'a pas de sens *a priori*. Dans *OpenMusic*, cet environnement permet d'organiser dans le temps un matériau sonore "hors-temps" issu de calculs effectués dans les *Patchs*.

La représentation graphique des *Maquettes*, leur interface utilisateur et les concepts de composition sur lesquels elles reposent étaient idéales pour l'implémentation de notre modèle.

Le formalisme sur lequel nous nous appuyons dans cette implémentation correspond au premier modèle de partitions interactives que nous ayons écrits. Directement inspiré par *Boxes*, il comporte les caractéristiques suivantes :

- la hiérarchie est limitée à un seul niveau et la racine est de type linéaire, et son comportement temporel est le *point d'orgue* défini à la section 7.3.2.
- les points de contrôle des objets sont limités à leur début et leur fin.
- les processus associés aux objets n'ont que deux étapes de calcul (début et fin), de plus ils sont considérés comme autonomes et aucun branchement n'est possible entre eux.
- les relations temporelles entre les objets sont des relations d'intervalles.
- tous les intervalles sont *souples*.
- la seule contrainte globale définissable est la limitation du nombre d'objets se déroulant simultanément.

Notre utilisation de relations d'intervalles mérite que l'on présente ici l'ensemble de relations que nous mettons à la disposition du compositeur.

Partant des travaux réalisés dans *Boxes*, dans lequel une seule relation est définissable entre deux boîtes, les relations proposée étant celles utilisées par Allen [4], nous décidâmes dans un premier temps de n'autoriser la définition que d'une seule relation entre deux objets. Pour ne pas trop restreindre le pouvoir d'expressivité des compositeurs, nous modifiâmes alors les relations proposées dans *Boxes* pour les rendre moins strictes. Nous présentons ces relations, qui gardent les mêmes noms que dans les travaux de Allen, sur les figures 10.1 et 10.2 en proposant leur équivalence avec des ensembles de relations de précédence entre les débuts et fins d'objets. Nous donnons également leur expression sous forme de S-mots.

Dans ce contexte, l'algorithme de compilation des partitions en réseaux de Petri n'est guère différent de celui présenté dans la partie précédente pour le modèle de base. Les relations de Allen étant considérées comme des conjonctions de relations temporelles *Pre*, il suffit d'effectuer les opérations de fusion et de mise en causalité équivalente à chaque relation.

Sur la base de ce modèle, nous avons développé dans les maquettes un outil d'édition de partitions interactives, ainsi que l'architecture de machine *ECO* correspondante. On trouvera dans [6] les détails de l'implémentation de la machine *ECO* pour ce formalisme.

Édition

La modification des maquettes pour permettre l'édition de partitions interactives a consisté, dans un premier temps, à permettre la définition de relations de Allen entre objets. Cette possibilité s'est accompagnée de l'implémentation de fonctionnalités chargées de maintenir ces relations tout au long de l'édition d'une partition par le compositeur. Ces fonctionnalités impliquant la résolution d'un système de contraintes lorsque l'utilisateur ajoute une relation ou modifie les dates d'un objet, s'appuient sur la librairie de résolution *Gecode*[21] ([90]). Ceci permet au système de propager une modification d'une date de début ou de fin d'un objet.

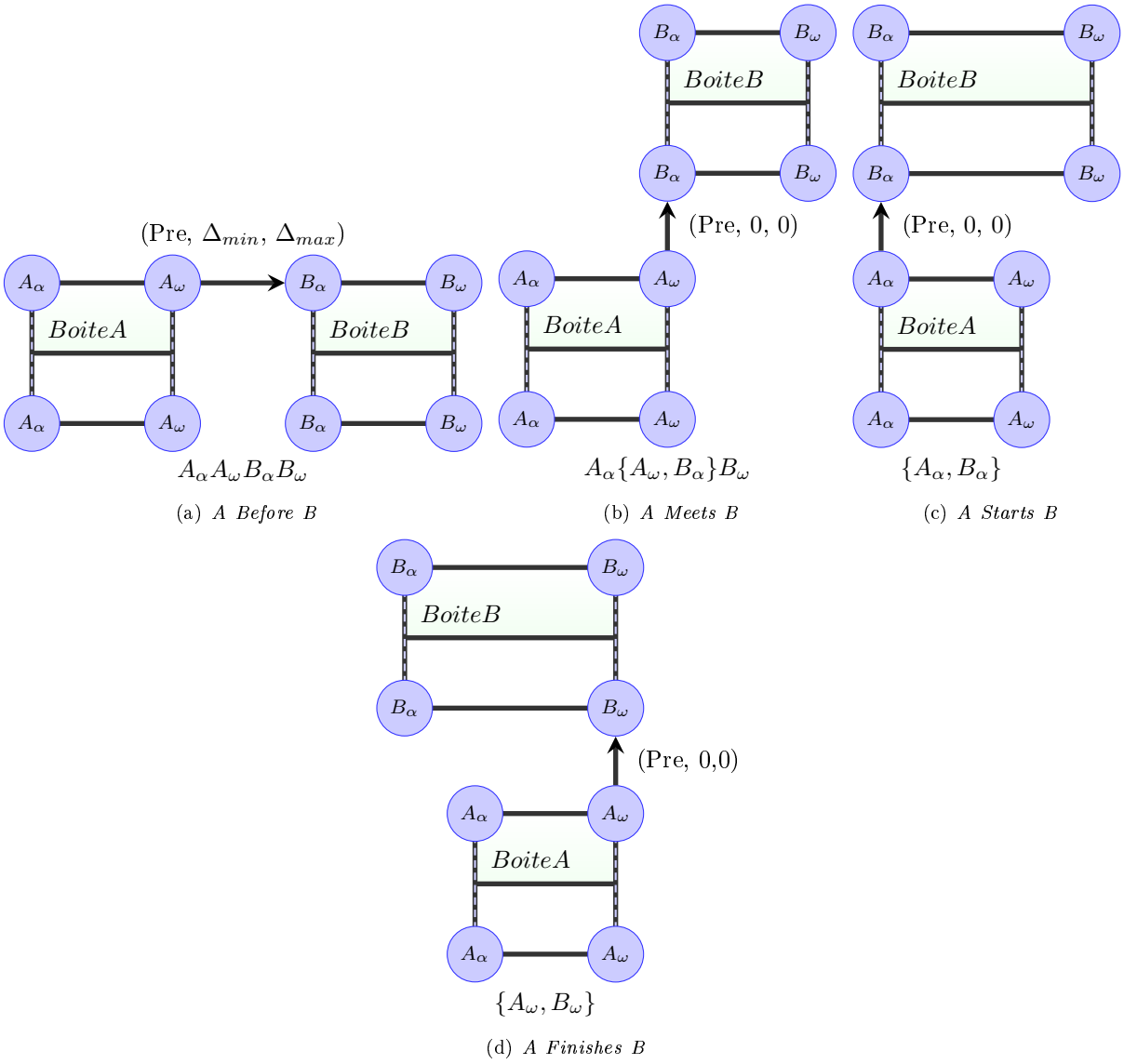
De plus, nous avons choisi dans cette implémentation d'autoriser la définition de relations temporelles non vérifiées au moment de leur introduction dans la partition. Le système cherche alors à résoudre le système de contraintes en y incluant la nouvelle relation. Si une solution est trouvée, les dates des objets sont alors modifiées. Si aucune solution n'est trouvée, le système en informe l'utilisateur.

L'outil d'édition permet également l'introduction de points d'interaction sur les débuts et fins d'objets. Aucune vérification n'est opérée à leur sujet. Les processus ne sont pas pris en charge directement dans *OpenMusic*, en effet, les patchs de ce dernier sont prévus pour effectuer des calculs en statique et ne peuvent être directement impliqués dans des exécutions dynamiques. Par conséquent, nous utilisons des applications tierces pour exécuter les calculs des processus. Ceux-ci sont supposés ne pas présenter de contraintes intrinsèques et leur utilisation est sous l'entière responsabilité du compositeur. Les processus propres aux partitions (déclenchés par la machine *ECO*) consistent à envoyer à l'application extérieure chargée du calcul, un message lorsque le processus débute et un autre lorsqu'il se termine.

La figure 10.3 présente un schéma du fonctionnement du système couplé avec d'autres applications (*Max/MSP*) sur cet exemple.

Pour effectuer cette communication entre *OpenMusic* et les applications extérieures, nous nous appuyons sur le protocole OSC[22]. L'outil d'édition dispose donc d'une interface permettant la définition des messages associés aux débuts et fins d'objets comme en témoigne la figure 10.4. Les messages associés à un objet ne sont pas modifiables en cours d'exécution. Les points d'interactions sont eux-mêmes conditionnés à la réception de messages OSC. La communication OSC s'effectue donc dans les deux sens.

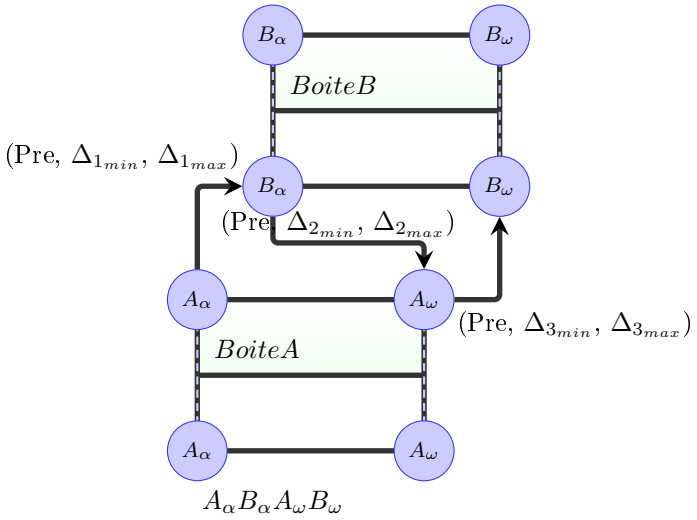
[21] www.gecode.org
[22] Open Sound Control (http ://opensoundcontrol.org/)



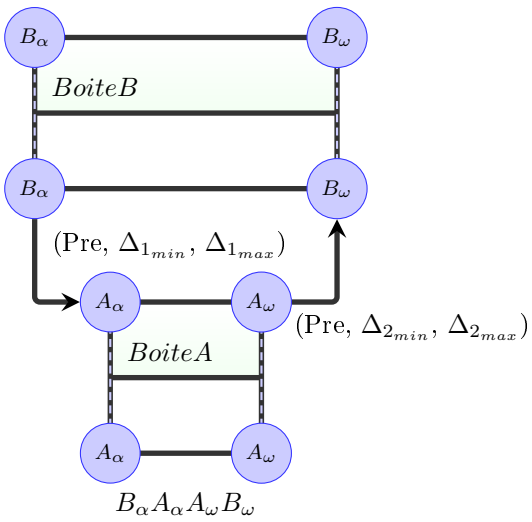
(a) *A Before B* (b) *A Meets B* (c) *A Starts B*

(d) *A Finishes B*

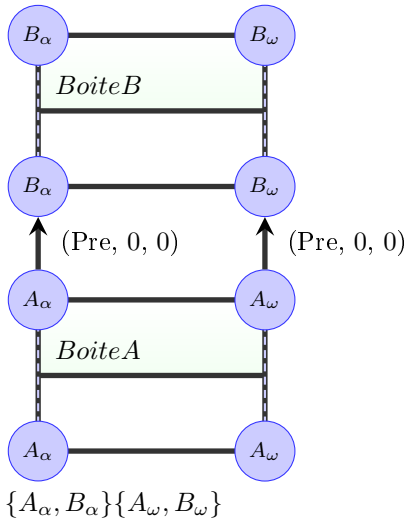
FIG. 10.1 – Relations de Allen "modifiées" sous forme de conjonctions de *Pre*



(a) *A Overlaps B*



(b) *A During B*



(c) *A Equals B*

FIG. 10.2 – Relations de Allen “modifiées” sous forme de conjonctions de *Pre* (suite)

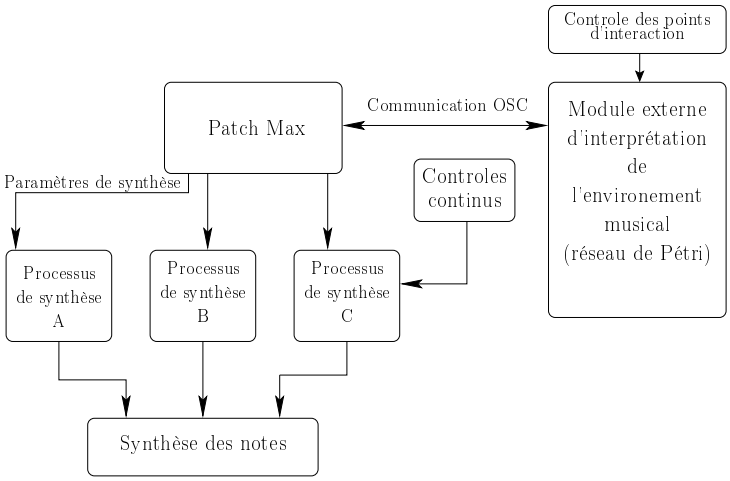


FIG. 10.3 – Schéma de fonctionnement du système

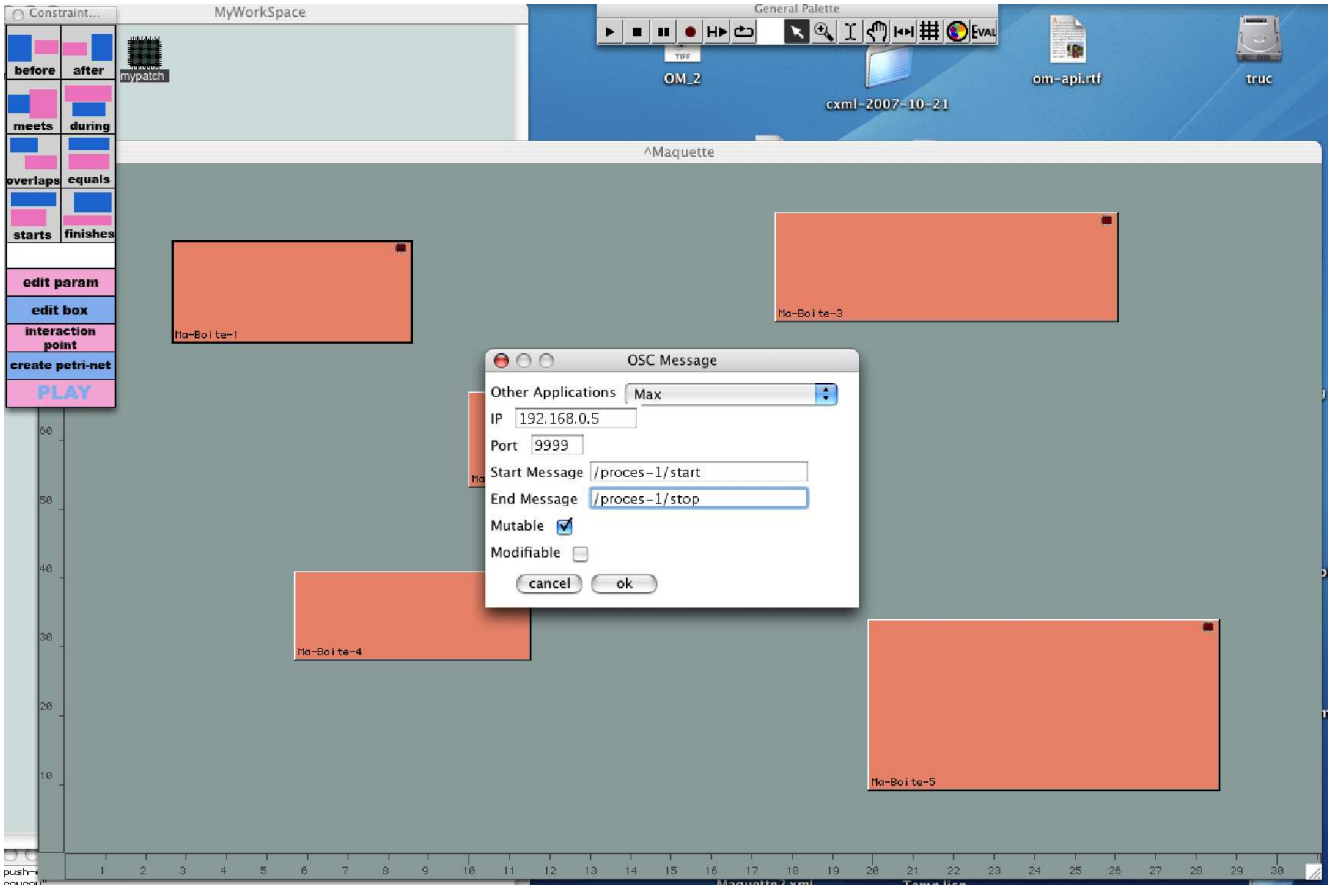


FIG. 10.4 – Capture d'écran de l'interface de définition des messages OSC dans *OpenMusic*

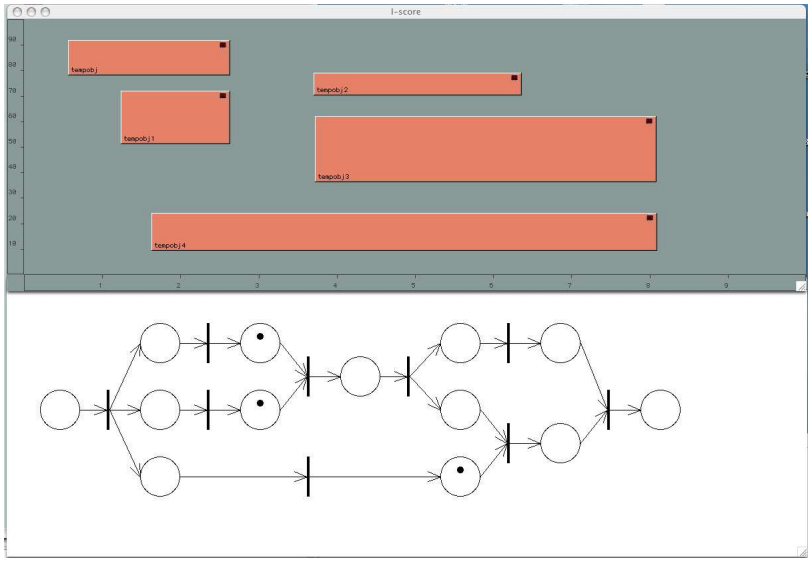


FIG. 10.5 – Une partition et son réseau de Petri dans *OpenMusic*. Le réseau de Petri est ici présenté car c'est pour le moment l'interface graphique à disposition de l'utilisateur pendant l'exécution. Les transtions du réseau change de couleur au cours de l'exécution selon si elles sont déclenchables ou non. L'évolution du réseau est également traduite par la consommation et la production des jetons.

On peut remarquer dans la boîte de dialogue permettant de donner les caractéristiques d'un objet, des onglets *mutable* et *modifiable*. Ces deux onglets sont en lien avec la contrainte globale limitant le nombre d'objets exécutables simultanément. Ils permettent de définir la réaction du système si au cours de l'exécution, le déclenchement d'un objet venait à rompre la contrainte sur le nombre d'objets exécutables. Si aucun onglet n'est coché, le déclenchement de l'objet est rétardé jusqu'à qu'il devienne possible, comme les intervalles sont tous souples, cela ne pose pas de problèmes pour le respect des relations temporelles. Si l'option *mutable* est choisie, l'objet est déclenché mais ses messages ne sont pas envoyés. L'option *modifiable* n'est pas implémentée mais anticipe les contraintes globales sur le contenu des objets (les entrées/sorties). Si le déclenchement d'un objet avec ses valeurs de sortie courantes invalide une contrainte de contenu, en choisissant cette option, le compositeur autorisera le système à calculer de nouvelles valeurs de sortie respectant la contrainte. Il est alors nécessaire de modifier les messages *OSC* associés à l'objet au cours de l'exécution.

Une fois la partition terminée, l'utilisateur peut déclencher sa compilation en vue de son exécution.

Exécution

Nous l'avons déjà évoqué, la transformation en environnement exécutable par la machine *ECO* est du même ordre que celle du modèle de base présenté précédemment. L'ordonnanceur de la machine est chargé d'exécuter le réseau issu de la compilation, tandis que le processeur contient et régit les processus envoyant les messages. On peut observer que dans ce cas particulier, tous les processus sont synchrones avec la machine *ECO* puisqu'ils n'ont pas d'étapes intermédiaires.

L'interface d'exécution est tout simplement l'affichage du réseau de Petri. Les transitions représentant des événements interactifs sont présentées dans une couleur différente des transitions statiques. Lorsqu'une transition interactive est déclenchable, l'utilisateur en est informé par un changement de couleur. La figure 10.5 présente une partition dans *OpenMusic* et son réseau de Petri tel qu'il apparaît.

Une transition du réseau est dessinée à la même ordonnée que l'objet dont elle représente le début ou la fin. Naturellement sur l'exemple proposé, les relations de Allen entre les objets ont impliquées des opérations de fusion pendant la compilation, l'ordonnée d'affichage des transitions est alors celle d'un des deux objets.

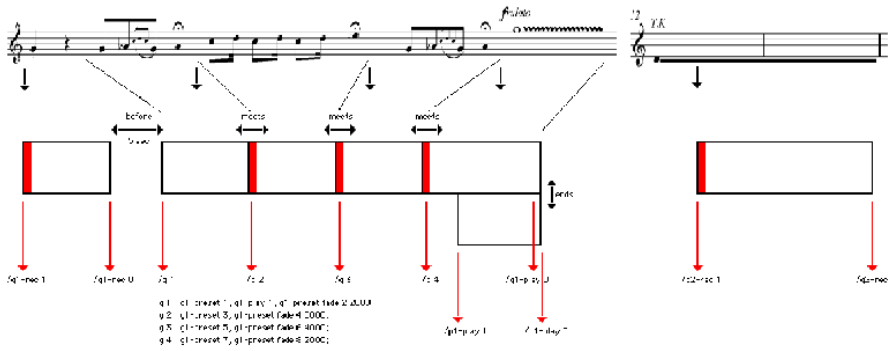


FIG. 10.6 – Un extrait de la partition *Le Pays du Soleil Couchant*

FIG. 10.7 – Capture d'écran de *Iscore*

10.1.2 Électro-acoustique

Cette implémentation est destinée à la composition et l'interprétation de pièces électro-acoustiques. Mais, le développement de l'interface spécifique pour les partitions interactives dans *OpenMusic*, n'a pas permis d'aboutir à une interface réellement utilisable par un compositeur seul face à l'outil.

Cependant, nous avons pu collaborer avec Joseph Larralde, assistant musical au Scrime[23], qui a composé une pièce mixte pour flûte et ordinateur intitulé *Le Pays du soleil Couchant* en s'appuyant sur notre modèle de partitions interactives.

La figure 10.6 présente la partition de cette pièce qui fait également intervenir de la synthèse et des traitements sonores utilisant *MAX/MSP*.

Lors des différentes présentations que nous avons faites de ces travaux, plusieurs compositeurs se sont montrés intéressés pour utiliser notre formalisme et les outils afférents. De nouvelles collaborations seront donc envisageables lorsque le développement du logiciel aura avancé.

On peut noter qu'une implémentation indépendante nommée *Iscore* est actuellement développé au LaBRI par Luc Vercellin en collaboration avec Jean-Louis Di Santo. Une attention toute particulière est apportée dans cette implémentation à l'apparence des objets des partitions, l'objectif étant de proposer une notation graphique des partitions d'électro-acoustique. La figure 10.7 présente une capture d'écran de *Iscore*.

10.1.3 Format XML pour la musique interactive

Le développement de l'outil d'édition de partitions nous a conduit à proposer un format de sauvegarde. Un premier constat nous a montré une certaine proximité entre le format que cherchions à élaborer, et des formats existants de documents multimédias comme *Hytime*[24] ([59]) ou *SMIL*[25] ([91]), et utiliser l'un d'eux aurait pu constituer une solution ; mais comme nous l'avons montré dans [5], notre format constituait une approche originale de par son application musicale, là où les formats existants ne s'intéressaient qu'à des documents non spécifiques. Nous avons donc écrit un format *XML* appelé *IscoreXML*, pour représenter des partitions écrites dans le formalisme utilisé pour l'implémentation dans *OpenMusic* (relations de Allen).

[23] Studio de Création et de Recherche en Informatique et musique Électro-acoustique
[24] Hypermedia/Time-based Structured Language
[25] Synchronized Multimedia Integration Language www.w3.org/AudioVideo/

FIG. 10.8 – IscoreXML comme interface avec d'autres formats

L'intérêt d'utiliser une syntaxe XML est de faire passer notre format de la simple sauvegarde de partitions, à l'interface avec d'autres formats. La figure 10.8 résume les conversions que nous envisageons.

L'exportation vers *SVG*[26] vise la diffusion de représentation des partitions et leur impression. L'interface avec *SMIL* a pour but la diffusion de simulations des pièces. Ces simulations ne pourront bien sûr accepter que les fonctionnalités entrants dans la norme SMIL. L'interaction avec ces simulations sera donc assez réduite.

L'importation depuis des fichiers *MusicXML*[27] est la seule que nous ayons commencé à développer. *MusicXML* est un format d'encodage de la musique instrumentale essentiellement tourné vers la notation occidentale "classique" mais qui tend à intégrer d'autres notations (notation du Moyen Âge). Les nombreuses possibilités qu'il offre l'ont peu à peu imposé comme un standard de fait. C'est ainsi qu'il est supporté par un grand nombre d'applications musicales.

La transformation d'un fichier *MusicXML* dans notre format permet l'importation automatique de partitions instrumentales dans l'outil d'édition. Pour qui voudrait effectuer cette importation "à la main", la création d'un objet par note s'avérerait rapidement fastidieuse. Nous avons donc partiellement élaboré et implémenté un algorithme qui analyse une partition *MusicXML* pour déduire les relations temporelles qu'elle implique.

A partir d'un *tempo* de base fourni par l'utilisateur, le système transforme la notation rythmique de la partition instrumentale en objets musicaux, à raison d'un objet par note. Puis les relations d'intervalles entre les notes sont ajoutées, elles permettront dans le cas d'une exécution interactive de maintenir l'organisation des notes : synchronisations entre plusieurs voix, succession des notes sur une même ligne mélodique. Des options permettent à l'utilisateur de fournir suffisamment d'informations pour que le système puisse instancier automatiquement les messages *OSC* associés aux objets.

Une fois une partition importée dans l'outil d'édition, il est possible d'y placer des points d'interaction pour interpréter la pièce dont les notes seront synthétisées par une application externe.

A titre d'exemple, la figure 10.9 présente une transcription des premières mesures du *Quintette pour Clarinette et Cordes K. 581* de Mozart, on peut remarquer sur cette figure qu'une importation manuelle d'une pièce instrumentale placerait l'utilisateur dans la peau de Sisyphe.

[26] Scalable Vector Graphics www.w3.org/Graphics/SVG/
[27] http ://www.recordare.com/xml.html

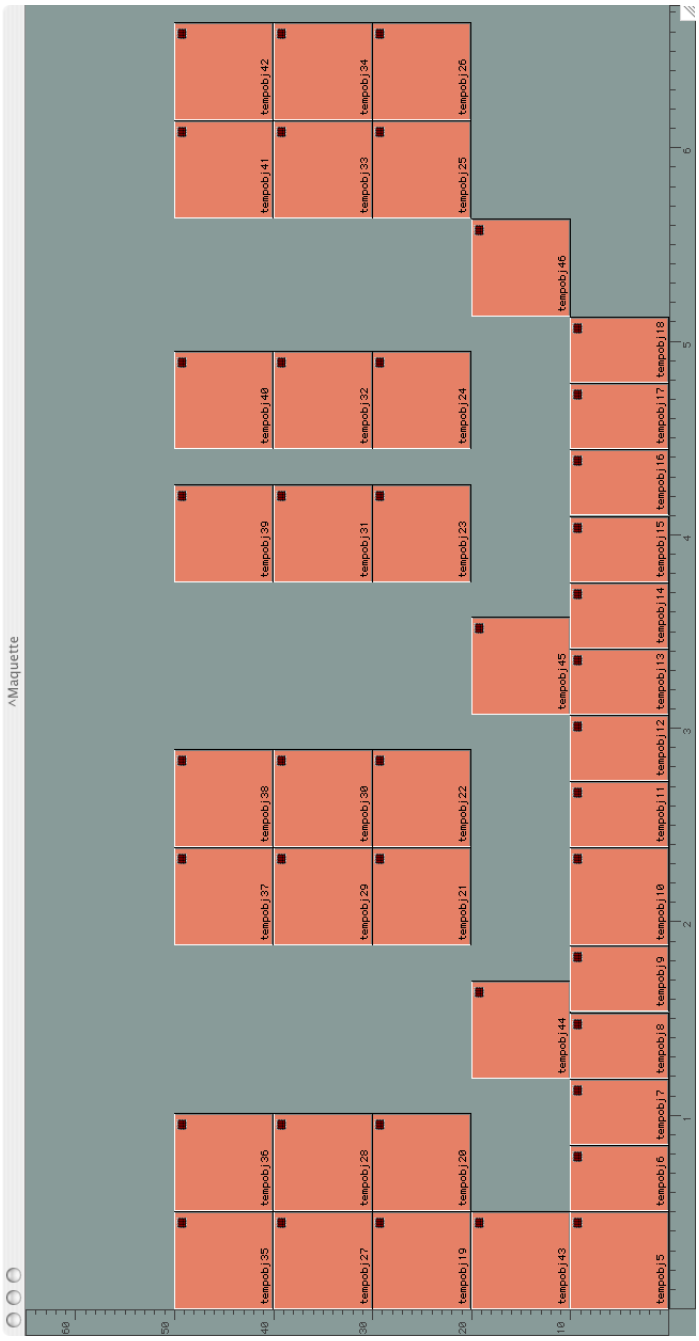


FIG. 10.9 – Premières Mesures du Quintette K. 581

Ces importations rendent possibles plusieurs applications du système de partitions interactives.

Tout d'abord, un système de play-back suivant le musicien. Il suffit pour cela que l'exécutant puisse déclencher les points d'interaction de la partition pendant qu'il joue.

Une autre application est la possibilité d'expérimenter l'interprétation d'une pièce instrumentale sans être capable de l'exécuter physiquement, à cause d'une maîtrise technique insuffisante ou tout simplement parce qu'elle comporte plusieurs instruments. Outre les applications pédagogiques que nous présentons par la suite, cette approche permettrait de proposer des dispositifs de jeu à des personnes handicapées sur le modèle du *Bao Pao* ([53]). L'intérêt de notre formalisme étant ici la création automatique de la partition depuis un fichier *MusicXML*.

Enfin, on peut imaginer que notre format XML, en intégrant les apports du formalisme complet, puisse être un format général de représentation de la musique et de l'interaction avec elle. Pour aboutir cette perspective implique d'intégrer des représentations XML des calculs effectués par les processus.

10.1.4 Pédagogie

Les applications pédagogiques ne manquent pas pour notre modèle surtout si il permet l'importation de partitions instrumentales. En effet, la possibilité offerte de sentir les interprétations possibles d'une partition sans être techniquement capable de la jouer, ouvre de nouvelles perspectives pour la pédagogie musicale. Ces applications pourraient aller en direction de musiciens travaillant une œuvre, ou peut-être avec plus de succès vers des non musiciens.

Dans ce cadre essentiellement centré sur la musique instrumentale, une interface basé sur objets temporels à la forme de boite ne semble guère adaptée. Nous avons donc entamé une collaboration avec Robert Strandh autour de son logiciel de notation musicale *Gsharp* [85]. Ce dernier permet de produire des partitions instrumentales de bonne qualité graphique, respectant les règles de gravure des éditions artisanales, et également de "jouer" une partition notée gâce à une communication *Midi* vers un synthétiseur. L'objectif est d'intégrer à *Gsharp* la partie exécution de notre modèle. En permettant la définition de notes interactives (début ou fin interactive), *Gsharp* pourrait devenir un environnement de notation instrumentale permettant l'interprétation des partitions notées, toujours au moyen d'un synthétiseur. L'utilisation de la notation classique donnerait alors à *Gsharp* les atouts nécessaires aux applications pédagogiques envisagées.

Cet outil pourrait également permettre d'éditer aisément les partitions de Jean Haury pour un "clavier à deux touches", dont le formalisme vise à simplifier le geste musical ([54]). Un module de transformation depuis le format de ces partitons vers notre format XML permettrait de profiter du catalogue des œuvres déjà écrites dans ce formalisme.

10.2 *Virage* vers le théâtre

Des liens entre l'informatique musicale et d'autres arts numériques existent notamment au travers de l'AFIM[28] en France. Ces échanges ont permis notre participation au projet ANR[29] *Virage*[30] qui a pour objet la création d'un prototype de séquenceur interactif pour la régie numérique de spectacle vivant, ainsi que l'analyse de sa prise en main par les professionnels du théâtre.

Depuis quelques années maintenant, la création de spectacle implique la diffusion de contenus multimédias, de plus le travail de régie intègre de plus en plus l'outil informatique pour gérer les contenus "traditionnels" (son, lumières...). Le besoin d'un outil d'écriture de conduites utilisant ces médias hétérogènes s'est fait alors sentir. Une conduite est assez proche d'une partition puisqu'elle traduit l'organisation temporelle entre les éléments du spectacle.

De plus, le jeu fluctuant des acteurs sur scène nécessite une souplesse du temps au moment de l'exécution afin de pouvoir les suivre. Ces caractéristiques proches de notre problématique nous ont permis de proposer notre modèle comme base au séquenceur.

[28] Association Française d'Informatique Musicale www.afim-asso.org
[29] Agence Nationale pour la Recherche
[30] Virage www.virage-platform.org/

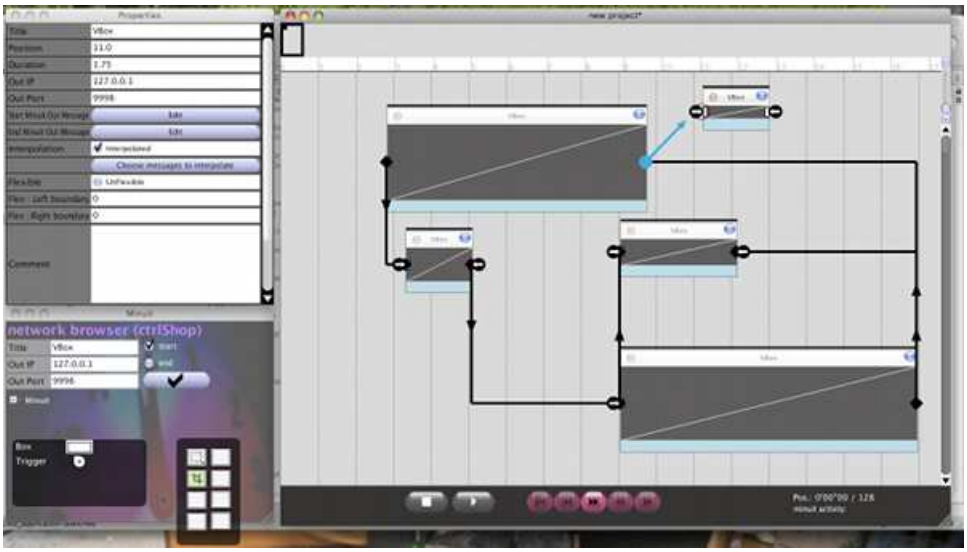


FIG. 10.10 – Une capture d'écran du séquenceur *Virage*

Les outils utilisés actuellement pour la régie numérique sont souvent issus du monde de l'informatique musicale. Les notions de processus ou encore d'événements se retrouvent donc dans les conduites de spectacles impliquant l'informatique. Cet état de fait a facilité la proposition de notre modèle.

Dans le cadre de ce projet, une implémentation spécifique a été réalisée. Proche de l'interface "classique" basée sur des boîtes, elle adopte cependant des choix propres. La figure 10.10 présente une capture d'écran de son outil d'édition.

Quelques adaptations du modèle ont été nécessaires pour permettre son utilisation dans ce cadre. Si le formalisme utilisé est très proche de notre formalisme complet de partitions interactives, un comportement d'intervalle spécifique a du être imaginé.

Stratégie de modification spécifique

Les régisseurs que nous avons rencontrés dans le cadre du projet *Virage*, ont souhaité voir adopter une stratégie de réaction à la modification de la date d'un point d'interaction que nous n'avions pas envisagée.

Cette stratégie consiste à modifier uniquement l'intervalle suivant immédiatement le point d'interaction, et à introduire un délai dans la suite si cet intervalle se trouve réduit à zéro.

Nous ne pouvons pas réellement parler de comportement d'intervalle pour cette stratégie dans la mesure où les régisseurs ont souhaité disposer de cette stratégie "par défaut", même en l'absence de relations temporelles limitant la durée des intervalles. S'agissant d'une approche par défaut, l'utilisation systématique d'un système de résolution de contraintes nous parut disproportionnée. Nous nous sommes donc appuyés sur le formalisme des réseaux de Petri à flux temporels pour répondre à ce problème. L'utilisation de la sémantique *Et-Maître* offre la possibilité d'intégrer cette stratégie directement dans le réseau. La figure 10.11 présente la portion de réseau de Petri correspondant à l'exemple précédent.

Le réseau d'occurence est construit de la même manière que pour le formalisme des partitions interactives et les arcs portent les intervalles de temps entre les événements tels qu'ils sont écrits sur la conduite. La prise en compte du pont d'interaction entraîne la création d'une place et d'arcs permettant de relier les événemements encadrant l'événement interactif ($end(A)$ sur l'exemple), et l'arc α porte la durée séparant $start(A)$ et $start(B)$ (Δ_{AB}). La transition $start(B)$ est alors définie avec la sémantique *Et-Maître* avec l'arc α comme maître.

Si on se souvient que la sémantique du *Et-Maître*, présentée dans la section 3.5.2.0 conduit à un intervalle de tir définit par :

$$MIN = \max_k \{\tau_k + \alpha_k\}$$

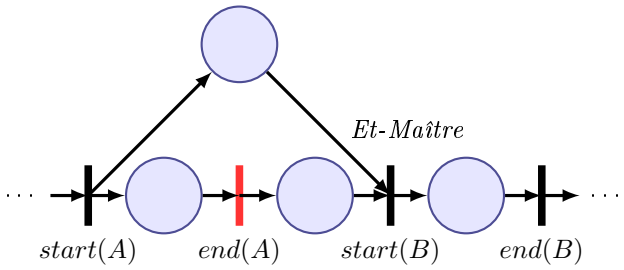


FIG. 10.11 – La configuration locale du réseau de Petri pour l'exemple de stratégie de modification du logiciel *Virage*

$$MAX = \min(\tau_i + \alpha_i, \max_k\{\tau_k + \alpha_k\})$$

avec i l'indice de l'arc maître, on peut observer que le calcul de MAX correspond exactement à la stratégie que l'on cherche à obtenir.

En effet, la durée Δ_{AB} devenant prioritaire elle sera respectée si $end(A)$ est trop retardé, ce qui aura pour effet de réduire l'intervalle entre $end(A)$ et $start(B)$. Si $end(A)$ est retardé de telle manière que Δ_{AB} ne puisse plus être respectée, alors $start(B)$ sera franchie dès que $end(A)$ le sera, ceci correspondnant bien à la stratégie souhaitée.

On peut également remarquer que la place ajoutée pour cette stratégie est une place implicite, qui ici fait sens. Au cours de la compilation, le réseau est obtenu comme précédemment puis ces places spécifiques et la sémantique *ET-Maître* sont ajoutées pour chaque point d'interaction.

Chapitre 11

Perspectives

La prévision est difficile, surtout lorsqu'elle concerne l'avenir.
Pierre Dac

11.1 Développements des travaux en cours

L'exposé que nous avons proposé présente l'état courant d'une thématique de recherche. Outre l'achévement de certains travaux débutés ici (aboutissement de la gestion des comportements, preuve des algorithmes), plusieurs pistes s'ouvrent pour le prolongement du sujet.

11.1.1 Homogénïsation du modèle temporel

La disctinction entre les modèles temporels des structures linéaires et des structures logiques restent un frein important pour un obtenir un modèle homogène. Ainsi, une amélioration majeure du modèle consisterait à prendre en compte les dates des points de contrôle d'une structure logique et de pouvoir définir des relations temporelles sur elles. Il faudrait alors être capable de raisoner temporellement avec des dates d'occurrences. Se posent alors un problème assez classique de véracité des relations temporelles entre certaines occurences des points de contrôles, mais entre toutes les occurences.

La représentation au travers d'un réseau de Petri, de partitions impliquant des relations temporelles entre des objets enfants d'une structure logique et ceux d'une structure logique semble ardue. En effet, en représentant "en dur" dans le réseau ce type de relations, un cas critique se produit lorsque les choix du musicien dans la structure logique font que des points de contrôle impliqués dans des relations avec des points de la structure linéaire, ne sont jamais déclenchés. L'attetente du déclenchement des points de la structure logique bloque le déroulement de la structure linéaire. Une solution serait éventuellement d'utiliser des réseaux auto-modifiant. Si à l'exécution, les choix du musiciens dans la structure logique empêche le déclenchement de points de contrôle impliqués dans des relations temporelles, les arcs qui représentent ces relations sont automatiquement supprimés pour éviter la situation de bloquage.

L'application des comportements dans ce contexte paraît assez complexe, dans la mesure où celle-ci est basée sur une analyse statique du réseau de Petri. Un réseau dynamique risque, par conséquent de poser quelques soucis. Une analyse des différentes configurations du réseau est éventuellement possible. Dans ces conditions, il faudrait pouvoir modifier le graphe de contraintes en même temps que le réseau au cours de l'exécution.

Pour complèter l'homogénéïté du modèle, il pourrait être intéressant de considérer les processus associés aux objets simples comme des réseaux de Petri, ce qui permettrait de les intégrer pleinement au modèle, et de pratiquer des analyses statiques plus poussées.

11.1.2 Optimisation de l'exécution du réseau de Petri

Pour l'instant l'exécution du réseau de Petri s'effectue par "interprétation", c'est à dire que les jetons sont consommés et produits dans les places exactement comme dans l'exécution théorique. Un voie pour

améliorer l'efficacité de l'exécution, serait de procéder à une compilation du réseau de Petri en vue de son exécution. Ceci consiste à identifier des parties du réseaux pour lesquelles la sémantique complète du tir des réseaux de Petri est superflue. Par exemple une portion "série" constituée d'un enchaînement de transitions statiques se ramène à un ensemble d'événement totalement ordonné, dont on connait les intervalles de temps séparant un intervalle de temps de son suivant. Ainsi, au moment de l'exécution, on peut calculer la date de déclenchement de chacun des événements à partir de celle du premier, il s'agit ensuite de simplement déclencher chaque événement à sa date calculée. Naturellement, il faut prévoir l'éventualité d'un changement de dates à la suite d'un écrasement, mais la propagation d'un jeton dans la branche, et la mise en œuvre d'un compteur pour chaque arc s'avèrent inutile.

11.1.3 Extensions du modèle courrant

Plusieurs évolutions du modèle actuel sont envisageables.

L'utilisation d'un outils de modélisation influençant l'approche que l'on a de l'objet à modéliser, il serait intéressant de tirer partie des jetons dans le réseau de Petri, pour proposer au compositeur la définition de contraintes portant sur le nombre de d'occurences possibles d'objets d'une structure logique.

La généralisation des contraintes d'intervalles à des combinaisons linéaires entre des intervalles quelconques et non plus simplement à des suites d'intervalles liés à des écrasements consituerait Un prolongement intéressant du modèle.

Enfin, une prise en compte plus spécifique de la musique rythmée permettrait aux compositeurs d'aborder d'autres univers musicaux. La composition de celle-ci demeure assez fastidieuse dans la mesure où la répétition d'un schéma rythmique sur une longue période, nécessite la reproduction des objets constituant ce schéma pour pourvoir les synchroniser avec des éléments d'une mélodie. La possibilité de synchronisation entre des éléments d'une structure logique et ceux d'une structure temporelle serait ici d'un grand secours. La capacité auto-modifiante du réseau serait utilisée ici, pour, après chaque occurence du schéma rythmique et l'exécution des objets mélodiques associé, synchroniser les éléments du schéma rythmique avec les éléments mélodiques associés à son occurrence suivante.

11.2 Nouvelles formes d'interaction

Un prolongement naturel de ce travail serait d'étendre le modèle aux autres possibilités d'interprétation. Pour ce faire, il s'agira de formaliser dans chaque cas, les libertés laissées à l'interprète et l'expression des limites imposées par le compositeur. La question de savoir si toute les formes d'interaction sont formalisables au travers de systèmes de contraintes n'est pas résolue, mais mais on peut espérer qu'elles le soient, tant l'écriture de l'interprétation se rattache à la définition d'ensemble de valeurs de paramètres acceptables.

Quoi qu'il en soit, l'*articulation* présente des caractères communs avec les variations agogiques comme nous le soulignons dans le premier chapitre. Elle devrait donc pouvoir s'exprimer à l'aide de relations temporelles telles que celles que nous proposons. Dans le cadre de la composition de musique électro-acoustique, le système actuel semble d'ailleurs convenir. Une étude sera toutefois nécessaire pour analyser les conséquences de l'articulation au niveau des paramètres des processus (par exemple des mixages différents si deux objets se chevauchent ou pas). En revanche, dans le cadre de l'application du système à la musique instrumentale (pédagogie, handicap...) une analyse des indications d'articluation permettrait d'instancier automatiquement les relations temporelles liées à ces indications. Comme différentes écoles d'instrummentistes ont donné des visions différentes de ces indications (durée de chevauchements par exemple), il serait possible de créeer au travers du format XML, un système de calques pour générer la partition interactive associée à l'interprétation d'un morceau selon telle ou telle école. Dans cette entreprise, une source d'informations intéressantes pourraient être le traité de tonotechnie (la manière dont on grave les cylindres des pianos mécaniques) du Père Angramelle [10], qui, pour noter la musique instrumentale au travers de la durée des notes, est amené à faire des choix concernant la position des notes selon l'interprétation voulue.

Parmi les développements des possibilités d'interaction, il en est un, lié au temps et particulièrement intéressant, mais que nous n'avons pas eu le temps d'abrder. C'est la possibilité de de modifier la vitesse

d'exécution d'un objet temporel au travers d'un contrôle continu. Nous avons chercher à formaliser une notion de "segment d'interaction", consistant en un intervalle dont l'interprète peut modifier la vitesse d'avancement du temps. Nous avons reculé à développer ce thème ici devant les problèmes théoriques qu'il soulève. Cependant, ce type d'interaction attise grandement l'intéret dde certains utilisateurs du système.

11.3 NTCC et approches multi-agents

La tentative de modélisation des partitions au travers du langage *NTCC* évoquée au chapitre 9, n'a pas donné lieu à une implémentation. Cependant, cette opportunité n'est pas à négliger. En effet, nos craintes concernant la vitesse de calcul pourraient se voir contredire par l'évolution de la puissance de calcul des machines, et l'efficacité des nouvelles implémentations de NTCC. De plus, cette approche permettrait d'apporter plus de souplesse dans la définition des relations. Si la force du modèle par réseau de Petri réside dans la possibilité qu'il offre de faire une analyse en statique du système et l'efficacité de son exécution, sa rigidité se transforme en faiblesse lorsqu'il s'agit de définir contraintes dont on autorise la non vérification dans certains cas. Le modèle NTCC permet ce type d'approche, par disjonction de relations temporelles par exemple.

Un autre formalisme, dont NTCC nest pas très éloigné, est l'utilisation de systèmes dits *multi-agents* [49]. La prise en compte d'une sosphistication de plus en plus grande pour les processus dans le modèle, permet d'imaginer une approche dans laquelle ils jouent le rôle central. Les relations temporelles sont alors décrites comme des caractéristiques locales des processus, l'organisation globale du système n'ayant plus à être explicitée. Si ce point de vue offre une grande souplesse, il faut cependant conserver des structures partagées de plus haut niveau pour gérer les mécanismes d'écrasement.

Conclusion

Ce mémoire se veut un panorama le plus complet possible de l'état de nos travaux visant à la définition et l'élaboration d'un système de partitions interactives pour l'interprétation de pièces électro-acoustiques. Bien qu'il s'agisse d'une photographie à un instant donné d'un travail toujours en cours, il est possible d'en tirer plusieurs bilans.

Tout d'abord, de la vaste question de l'interprétation de pièces électro-acoustiques nous avons pu dégager une problématique précise, en nous appuyant sur une formalisation des possibilités d'interaction. Nous focalisant sur les variations agogiques, nous avons alors fait émerger la question théorique sous-jacente à cette problématique par une analyse des variations agogiques, de l'expression des libertés qui leur attachées et de leurs limites. L'approche consistant à percevoir l'interprétation comme la définition d'espaces de liberté assimilables à la spécification de problèmes de contraintes, semble pouvoir s'adapter à d'autres aspects de l'interprétation.

De l'analyse de l'écriture et de la manipulation des variations agogiques, nous avons élaboré un formalisme de description de partitions interactives pour l'interprétation. Ce formalisme, pensé comme un langage de spécification d'organisation temporelle de processus dynamique, se veut fondamentalement dédié à la composition musicale, pour rester proche des préoccupations des compositeurs et ainsi rester facilement manipulable par ces derniers. En outre ce langage reste suffisamment générique pour accepter des évolutions ou des raffinements. Cette volonté se retrouve dans l'architecture modulaire de la machine *ECO*, à même d'accueillir de nouveaux éléments pour répondre à de nouveaux besoins. L'utilisation de la propagation par contraintes pour définir les stratégies d'adaptation aux modifications dynamiques, nous a permis de généraliser les comportements classiques de point d'orgue et de changement de tempo, et ainsi de proposer de nouveaux comportements.

Pour la machine d'exécution, nous avons chercher à modéliser le déroulement des partitions. Celles-ci étant perçues comme des des organisations temporelles globales de processus, les réseaux de Petri se sont avérés l'outil adéquat. Partant de cette modélisation, nous avons fourni un modèle de machine *ECO* basé sur un interpréteur de réseaux de Petri, déclenchant les points de contrôle des processus lors du franchissement des transitions du réseau modélisant une partition. Pour prouver la validité de l'algorithme de compilation, nous avons établi une correspondance entre réseaux d'occurrences et S-langages. Les réseaux de Petri produits par la technique de compilation que nous avons proposée, permettent les comportement de point d'orgue et changement de tempo. Pour obtenir l'exécution avec les autres types de comportements, nous avons adjoint aux réseau de Petri, un système de propagation de contraintes. Le réseau de Petri modélisant une partition, permettant alors une analyse statique de la partition pour construire le graphe de contraintes nécessaire à l'utilisation de la technique de propagation. Tous les algorithmes mettant en œuvre cette solution complète ne sont pas prouvé, mais nous avons fourni plusieurs éléments pour aboutir. De même, nous avons proposé plusieurs pistes pour modéliser des partitions dans le cadre général du formalisme.

Les différentes implémentations de ces bases théoriques, dans *OpenMusic* ont permis de commencer à tester la validité de la démarche, et ont débouché sur de nouvelles applications en direction de la pédagogie et de la musique instrumentale, ou encore du théâtre.

Enfin, nous avons exposé plusieurs voies pour l'amélioration ou l'extension du système, ou pour envisager des solutions alternatives à la modélisation par réseaux de Petri.

Cinquième partie

Annexes

Annexe A

Formalisation des méthodes de modification d'intervalles

Nous présentons ici le calcul des valeurs d'intervalles des méthodes de propagation.

Rappelons qu'il s'agit de propager la modification d'une valeur de variable à travers une contrainte imposant la relation :

$$\Delta = \Sigma_i \Delta_i$$

Cette relation correspondant à une relation temporelle :

$$rt =< t, \sigma_1, \sigma_n, \Delta_{min}, \Delta_{max}) >$$

telle qu'il existe un ordre total entre des événements intermédiaires :

$$\forall i \in [|1, n-1|], rt_i =< t, \sigma_i; \sigma_{i+1}, \Delta_{i_{min}}, \Delta_{i_{max}} >$$

A.1 Modification du membre gauche (Δ)

Supposons que la valeur Δ soit modifée en Δ' avec :

$$\Delta' = \Delta + \delta$$

Naturellement, la nouvelle valeur Δ' est dans les limites imposées par la relation rt i.e :

$$\Delta_{min} - \Delta \leq \delta \leq \Delta_{max} - \Delta \tag{A.1}$$

Cette variation implique une modification des Δ_i en Δ'_i avec

$$\forall i \in [1, n-1],\ \Delta'_i = \Delta_i + \delta_i$$

Dans la suite, on note Δ_{extr} et $\Delta_{i_{extr}}$, les valeurs définies comme suit :

$$\begin{array}{lrcl} \text{Si } \delta < 0, & \Delta_{extr} & = & \Delta_{min} \\ & \Delta_{i_{extr}} & = & \Delta_{i_{min}} \\ \text{sinon,} & \Delta_{extr} & = & \Delta_{max} \\ & \Delta_{i_{extr}} & = & \Delta_{i_{max}} \end{array}$$

Modification chronologique

En notant k l'indice tel que :

$$\sum_{i=1}^{k-1} |\Delta_i - \Delta_{i_{extr}}| < |\delta| \leq \sum_{i=1}^{k} |\Delta_i - \Delta_{i_{extr}}|$$

Le calcul de Δ_i

$$\begin{cases} \forall i \in [1, k-1], & \delta_i & = & signe(\delta).|\Delta_i - \Delta_{i_{extr}}| \\ & \delta_k & = & signe(\delta).(|\delta| - \sum_{i=1}^{k-1} |\Delta_i - \Delta_{i_{extr}}|) \\ \forall i \in [k+1, n], & \delta_i & = & 0 \end{cases}$$

Modification antichronologique

En notant k l'indice tel que :

$$\sum_{i=k-1}^{n} |\Delta_i - \Delta_{i_{extr}}| \leq |\delta| < \sum_{i=k}^{n} |\Delta_i - \Delta_{i_{extr}}|$$

Le calcul des Δ_i s'exprime comme suit :

$$\begin{cases} \forall i \in [1, k-1], & \delta_i & = & 0 \\ & \delta_k & = & signe(\delta).(|\delta| - \sum_{i=k+1}^{n+1} |\Delta_i - \Delta_{i_{extr}}|) \\ \forall i \in [k+1, n], & \delta_i & = & signe(\delta).|\Delta_i - \Delta_{i_{extr}}| \end{cases}$$

Notons au passage que dans les deux cas de modification, l'existence de l'indice k est assurée par l'inéquation A.1.

Modification proportionnelle

Dans ce cas, chaque Δ_i est modifié proportionnellement à la modification de Δ

$$\Delta'_i = \Delta_i . \frac{\Delta'}{\Delta}$$

A.2 Modification du membre droit (Δ_i)

Dans ce cas, en notant Δ_j l'intervalle modifié on a :

$$\Delta'_j = \Delta_j + \delta_j$$

La modification δ_j va se trouver être "absorbée" par les Δ_i pour i entre $j+1$ et n. Les calculs permettant permettant de déduire les δ_i correspondants se retrouvent en remplaçant dans les formules précédentes δ par δ_j est les intervalles $[1, n]$ par $[j+1, n]$.

On note :

$$\Delta_{ant} = \sum_{i=1}^{j-1} \Delta_i$$

$$\Delta_{post} = \sum_{i=j+1}^{n} \Delta_i \tag{A.2}$$

avec :

$$\Delta_{post_{min}} \leq \Delta_{post} \leq \Delta_{post_{max}}$$

On donc :

$$\Delta = \Delta_{ant} + \Delta_j + \Delta_{post}$$

Soit :

$$\Delta_{ant} + \Delta_j = \Delta - \Delta_{post}$$

On suppose :

$$\Delta_{min} - \Delta_{post_{max}} - \Delta_{ant} \leq \Delta'_j \leq \Delta_{max} - \Delta_{post_{min}} - \Delta_{ant}$$

C'est-à-dire que la modification est possible au regard des propriétés des différents intervalles en présence.

Dans le cas chronologique, Δ_j va modifier les Δ_i composant Δ_{post} dans l'ordre chronologique, puis Δ si besoin est.

Pour la modification anti-chronologique, la modification de Δ_j va être propagé à Δ avant de l'être aux Δ_i de Δ_{post} dans l'ordre antichronologique.

Modification chronologique

Dans ce cas :

– Si $|\delta_j| \leq |\Delta_{post_{extr}} - \Delta_{post}|$:

$$\delta_{post} = -\delta_j$$

Le calcul des Δ_i s'effectue alors en propageant chronologiquement dans l'équation A.2, dans laquelle Δ_{post} est membre gauche.

– Sinon :

$$\begin{cases} \delta_{post} = -signe(\delta_j).|\Delta_{post_{extr}} - \Delta_{post}| \\ \delta = \delta_j - \delta_{post} \end{cases}$$

Modification antichronologique

Dans ces conditions :

– Si $|\delta_j| \leq |\Delta_{extr} - \Delta|$:

$$\delta = \delta_j$$

– Sinon :

$$\begin{cases} \delta = signe(\delta_j).|\Delta_{extr} - \Delta| \\ \delta_{post} = \delta - \delta_j \end{cases}$$

Naturellement, les Δ_i sont modifiés antichronologiquement en propageant dans A.2.

Modification proportionnelle

Dans ce cas, la valeur Δ n'est pas modifiée et :

$$\delta_{post} = \Delta - \delta_j$$

Dans ce cas, les Δ_i de Δ_{post} sont modifiés par propagation proportionnelle au travers de l'équation A.2. A la différence du cas de modification externe, Δ_{post} ne dispose pas de rapport de quanta, il faut modifier les Δ_i de la manière suivante :

$$\Delta'_i = \Delta_i . \frac{\Delta'_j}{\Delta_j}$$

Annexe B

Construction du sous-graphe *série-parallèle*

Nous présentons ici l'algorithme de construction d'un sous-graphe *série-parallèle* maximale d'un graphe bipolaire.

Comme nous nous appuyons ici sur la strucuture de graphe du réseau de Petri, nous utilisons dans ce chapitre le vocabulaire général des graphes, et présentons un algorithme générique pour les graphes.

Etant donné que les réseaux de Petri sur lesquels nous travaillons sont des réseaux d'occurrences, une place du réseau a une unique transition prédécesseur et une unique transtion successeur, et la place représente l'intervalle de temps séparant ces deux transitions.

La structure de graphe sous-jacente (parfois appelée *graphe d'événements*) sur laquelle nous appliquons notre algorithme est alors la suivante :

- les sommets sont les transitions du réseau.
- les arcs entre sommets correspondent aux enchaînements "arc-place-arc" du réseau de Petri et sont pondérés avec l'intervalle correspondant.

On peut remarquer de plus que la structure de réseau d'occurrences nous assure que le graphe sous-jacent est orienté, acyclique et *bipolaire*, c'est à dire qu'il dispose d'une source et d'un puit, et que chaque sommet se trouve sur un chemin menant de la source au puit. Cette propriété nous permet d'ecrire un algorithme spécifique basé sur un parcours en profondeur depuis la source, qui cherche à construire la décomposition *série-parallèle* du graphe et qui détecte les éléments du graphe qui ne peuvent s'intégrer à une telle décomposition.

Pour détecter des branches invalidant la propriété série-parallèle, l'algorithme étiquette les sommets du graphe avec des ensembles de labels.

B.0.1 Complexité

En terme de parcours dans le graphe, comme il s'agit d'un parcours en profondeur auquel s'ajoute le traitement des labels le long de la composition-série-courante, la complexité du parcours effectué par l'algorithme est linéaire par rapport au nombre de sommets du graphe.

Cependant, le traitement des ensembles de labels nécessitent des opérations supplémenataires. La linéarité de l'algorithme générale ne sera donc assurée que dans le cas où les opérations sur les ensembles de labels s'effectuent en temps constant. Sans que nous puissions en apporter la preuve, une représentation des ensembles de labels par vecteurs de bits, devrait assurer une complexité en $O(1)$ pour les opérations sur ces ensembles.

Ceci devrait nous assurer une complexité linéaire pour cet algorithme.

Algorithme 3 : sous-graphe-serie-para

Entrées : graphe
source=`source`(graphe);
puit=`puit`graphe;
`ajouter-label`(source,0);
`ajouter-label`(puit,0);
indice-compo-para-courante=0;
pile-chemin=`nouvelle-pile`();
`empiler`(source,pile-chemin);
pile-compo-para=`nouvelle-pile`;
`empiler`(0,pile-compo-para) ; label-courant=();
`ajouter`(0,label-courant);
compo-serie-courante=`nouvelle-pile`();
`avance-label`(pile-chemin,pile-compo-para,
    indice-compo-para-courante, label-courant,
    compo-serie-courante, *vrai*,*faux*);

Algorithme 4 : avance-label

```
si pile-chemin ≠ ∅ alors
    sommet-courant=tete(pile-chemin);
    arc=retourne-arc-non-marque(sommet-courant);
    si arc==NULL alors
        traitement-sommet-pour-reculer(sommet-courant,
            pile-compo-para, indice-compo-para-courante,
            label-courant, compo-serie-compo-serie-courante,
            serie-para);
        avance-label(queue(pile-chemin),pile-compo-para,
            tete(pile-compo-para),indice-courant,label-courant,
            compo-serie-courante, faux, serie-para);
    sinon
        traitement-sommet-pour-avancer(sommet-courant,
            compo-para, indice-compo-para-courante, serie-para);
        marquer(arc);
        si card(sommets-sortants(sommets-courant)) ≠ 1 alors
            indice-courant ← indice-courant + 1;
            empiler(indice-courant,pile-compo-para);
            ajouter(indice-courant,label-courant);
        fin
        nouveau-sommet=(destination(arc));
        si label(nouveau-sommet)==NULL alors
            avance-label(empiler(nouveau-sommet,pile-chemin),
                pile-compo-para,indice-compo-para-courante,
                label-courant,compo-serie-courante,vrai,faux);
        sinon
            si est-inclus(indice-compo-para-courante,
                label(nouveau-sommet)) alors
                tant que tete(compo-serie-courante) ≠ nouveau-sommet faire
                    supprimer-label-inferieur(label(sommet),indice-compo-para-courante);
                    depiler(compo-serie-courante);
                fin
                ajouter-label(nouveau-sommet,label-courant);
                avance-label(pile-chemin,pile-compo-para,
                    indice-compo-para-courante,indice-courant,
                    label-courant,compo-serie-courante,faux,vrai);
            sinon
                avance-label(pile-chemin,pile-compo-para,
                    indice-compo-para-courante,indice-courant,
                    label-courant,compo-serie-courante,faux,faux);
            fin
        fin
    fin
fin
```

Algorithme 5 : traitement-sommet-pour-reculer

```
Entrées : sommet-courant, pile-compo-para,
     indice-compo-para-courante, label-courant,
     compo-serie-courante,serie-para
si card(arcs-sortants(sommet-courant))==1 alors
    si serie-para alors
        ajouter-label(sommet-courant,label-courant);
        empiler(sommet-courant,compo-serie-courante);
    fin
sinon
    si serie-para alors
        pour chaque sommet dans compo-serie-courante faire
            supprimer-label-superieur(label(sommet),
                indice-compo-para-courante)
        fin
        ajouter-label(sommet-courant,label-courant);
        empiler(sommet-courant,compo-serie-courante);
    fin
    depiler(pile-compo-para);
fin
```

Algorithme 6 : traitement-sommet-pour-avancer

```
Entrées : sommet-courant, pile-compo-para,
     indice-compo-para-courante, label-courant
si avance==faux alors
    depiler(pile-compo-para);
    si serie-para alors
        ajouter-label(sommet-courant,label-courant);
        indice-compo-para-courante ← tete(pile-compo-para);
        label-courant ← ();
        inserer(indice-compo-para-courante,label-courant);
    fin
fin
```

Annexe C

Démonstration des résultats sur les réseaux d'occurences et les S-langages

C.0.2 Opération de fusion

Preuve C.1 *Montrons tout d'abord que PN' est bien un réseau d'occurrences.*

Le nombre de prédécesseurs d'une place de PN' est borné par construction, supposer le contraire implique la contradiction de cette propriété pour PN.

Pour prouver la quasi-vivacité de PN', montrons qu'un marquage accessible sensibilise t_{12}. Comme PN est quasi-vivace :

$$\exists \sigma_1, \sigma_2 \in T, m_1, m_2 \in A(R, m_0) \ t.q. \ : \begin{array}{l} m_1[t_1>, \\ m_2[t_2>, \\ m_0[\sigma_1 > m_1, \\ m_0[\sigma_2 > m_2 \end{array}$$

On définit le marquage m_{12} dans PN' :

$$\forall p \in P, m_{12}(p) = m_1(p) \vee m_2(p)$$

Comme t_1 et t_2 sont en concurrence dans PN alors m_{12} est accessible dans PN'. De plus, par construction de PN', m_{12} sensibilise t_{12} dans PN'.

La quasi-vivacité de PN et l'accessibilité de t_{12} dans PN' permettent de montrer la quasi-vivacité de PN'.

PN' est donc un réseau d'occurrences.

Montrons la propriété sur les S-langages de réseaux.

Soit $\omega \in L$ $\mathbf{J}$ $[\{l(t_1, l(t_2)\}]$.

On a $\omega = l(\sigma)$.

Comme t_1 et t_2 sont en concurence, σ correspond à une séquence de franchissement dans PN au cours de la laquelle t_1 et t_2 sont tirées simultanément, ou à une séquence au cours de laquelle ni l'une ni l'autre n'est tirée. Dans ce dernier cas, σ est clairement une séquence de franchissements dans PN', la structure de PN' étant la même que celle de PN en dehors de la transition t_{12}. Et comme par construction :

$$l'(\sigma) = l(\sigma)$$

par définition de $Term'$, $l'(\sigma) \in L'$ d'où :

$$\omega \in L'$$

Dans l'autre cas, on a :

$$\sigma = \sigma_{ant}\{t_1, t_2, \ldots, t_n\}\sigma_{post}$$

L'ensemble de transitions tirées simultanément à t_1 et t_2 ne se réduit pas nécessairement à ces deux transitions.

On a alors :

$$m_0[\sigma_{ant} > m_{ant}$$

Le marquage m_{ant} sensibilise t_1 et t_2 dans PN. Mais m_{ant} sensibilise t_{12} dans PN', d'après la définition de la fonction Pre'. En outre, en notant σ' tel que :

$$\sigma' = \sigma_{ant}\{t_{12}, \ldots, t_n\}\sigma_{post}$$

d'aprés les définition de $Post'$ et de $Term'$,

$$l'(\sigma') \in L'$$

or d'après la définition de l', $l'(\sigma') = \omega$, d'où :

$$\omega \in L'$$

On a donc :

$$L \mathbf{J} [\{l(t_1, l(t_2)\}] \subset L'$$

Soit $\omega' \in L'$.

On a $\omega' = l'(\sigma')$.

Par des considérations sur la structure de PN', σ' montre que σ' est une séquence de franchissements, et grâce aux définitions de l' et $Term'$ on a :

$$l(\sigma') \in L$$

et

$$l(\sigma') = l'(\sigma') = \omega'$$

d'où :

$$\omega' \in L$$

De plus, grâce à l'uncité de la représentation des événements par l qui se répercute à l' :

$$\omega'_{|\alpha_{12}} = \{l(t_1), l(t_2)\}$$

donc

$$\omega' \in L \mathbf{J} [\{l(t_1, l(t_2)\}]$$

et

$$L' \subset L \mathbf{J} [\{l(t_1, l(t_2)\}]$$

Finalement :

$$L' = L \mathbf{J} [\{l(t_1, l(t_2)\}]$$

C.0.3 Opération de causalité

Preuve C.2 *Montrer que PN' est un réseau d'occurrences revient à prouver la sensibilisation de t_2 dans PN'. Ceci est assuré par le fait d'une part que PN est un réseau d'occurrences, que t_1 et t_2 ne sont pas en conflit et que t_2 ne précède pas t_1. Dans ces conditions, le franchissement de t_1 n'est pas conditionné par celui de t_2, la place créee lors de la transformation recevra donc un jeton et s'intégrera à un marquage sensibilisant t_2.*

La démonstration de la propriété sur les S-langages est semblable à celle de la transformation par fusion. Elle consiste donc à montrer la double inclusion en analysant les séquences de franchissements et en exhibant des marquages itermédiaires. Par soucis de concision, elle n'est pas développée ici.

Table des figures

Liste des Algorithmes

Bibliographie

[1] A.Friberg and J. Sundberg. Time discrimination in a monotonic, isochronous sequence. *The Journal of the Acoustical Society of America*, 98(5) :2524–2531, 1995.

[2] C. Agon. *OpenMusic : Un langage visuel pour la composition musi-cale assistée par ordinateur.* PhD thesis, Université Paris 6, Pierre et Marie Curie, 1998.

[3] Y-K. Ahn, C. Agon, and M. Andreatta. Vers un module d'analyse computationnelle sous openmusic. In *Actes des* 13^{emes} *journées d'Informatique Musicale (JIM08), Albi*, 2008.

[4] J.F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11) :832–843, 1983.

[5] A. Allombert. Un format de partitions interactives. *Documents Numériques*, 11(3-4) :29–44, 2008.

[6] A. Allombert, G. Assayag, and M. Desainte-Catherine. A system of interactive scores based on petri nets. In *Pr. of the 4th Sound and Music Computing Conference (2007), Lefkada, Greece*, July 2007.

[7] A. Allombert, G. Assayag, M. Dessainte-Catherine, and C. Rueda. Concurrent constraints models for interactive scores. In *Pr. of the 3rd Sound and Music Computing Conference (2006), GMEM, Marseille, France*, May 2006.

[8] S. Alstrup, C. Gavoille, H. Kaplan, and T. Rauhe. Nearest common ancestors : A survey and a new algorithm for a distributed environment. *Theory of Computing Systems*, 37(3), 2004.

[9] M. Andreatta. *Méthodes algébriques en musique et musicologie du* XX^e *siècle aspects théoriques, analytiques et compositionnels.* PhD thesis, Ecole des Hautes Etudes en Sciences Sociales, 2003.

[10] J. Angramelle. *Le tonotechnie ou l'art de noter les cylindres.* Minkoff, 1971, 1775.

[11] G. Assayag. Cao : Vers la partition potentielle. *Les cahiers de l'Ircam - La composition assistée par ordinateur*, 1993.

[12] G. Assayag, C. Agon, J. Fineberg, and P. Hanappe. Problèmes de notation dans la composition assistée par ordinateur. In *Actes des rencontres pluridisciplinaires musique et notation, Grame, Lyon, France*, 1997.

[13] G. J. Badros and A. Borning. The Cassowary linear arithmetic constraint solving algorithm : Interface and implementation. Technical Report UW-CSE-98-06-04, University of Washington, `http ://www.cs.washington.edu/research/ constraints/cassowary`, 1998.

[14] M. Balaban and Murray N. Interleaving time and structure. *Computers and Artificial Intelligence*, 17(1) :1–34, 1998.

[15] G. Bennet. Repères électro-acoustiques. *Contrechamps*, 11 :29–52, 1990.

[16] A. Beurivé and M. Desainte-Catherine. Representing musical hierarchies with constraints. In *Proceedings of CP'01, Musical Constraints Workshop*, 2001.

[17] A. Bonnet and C. Rueda. Situation : un logiciel de programmation par contraintes pour l'aide à la composition musicale. In *Proc. of the JIM'94 (Journées d'informatique Musicale), Bordeaux*, pages 1–4, March 1994.

[18] A. Borning, R. Lin, and K. Marriot. Constraints for the web. In *Proc. of the* 5^{th} *ACM internation Multimedia Conference*, 1997.

[19] P. Boulez. *Penser la musique aujourd'hui.* Gonthier, 1963.

[20] P. Boulez. *Jalons (dix ans d'enseignement au Collège de France).* Jean-Jacques Nattiez, 1989.

[21] R. Brachman and J.Schmolze. An overview of the kl-one knowledge representation system. *Cognitive Science*, 9, 1985.

[22] J. Bresson. *La synthèse sonore en composition musicale assistée par ordinateur. Modélisation et écriture du son.* PhD thesis, Université Paris 6, Pierre et Marie Curie, 2007.

[23] A. Camurri, A. Catorcini, C. Innocenti, and A. Massari. Music and multimedia knowledge and reasoning : the harp system. *Computer Music Journal*, 19(2) :34–58, 1995.

[24] A. Camurri, G. Haus, and R. Zaccaria. Describing and performing musical processes by means of petri nets. *Interface*, 15 :1–23, 1986.

[25] G. Carpentier. *Approche computationnelle de l'orchestration musicale. Optimisation multicritère sous contraintes de combinaisons instrumentales dans de grandes banques de sons.* PhD thesis, Université Paris VI, Pierre et Marie Curie, 2008.

[26] J. Chadabe. Interactive composing : an overview. *Computer Music Journal*, 8(1), 1984.

[27] Eric C. Clarke, R. Parncutt, M. Raekallio, and J. A. Sloboda. Talking fingers : An interview study of pianists views on fingering. *Musicae Scientae*, 1(1) :87–107, 1997.

[28] P. Codognet and D. Diaz. Yet another local search method for constraint solving. *LNCS - 1^{st} Symposium on Stochastic Algorithms : Foundations and Applications*, 2246, 2001.

[29] T. Coduys and G. Ferry. Iannix - aesthetical/symbolic visualisations for hypermedia composition. In *Proc. of the 1^{st} Sound and Music Computing Conference (SMC04), Paris, France*, 2004.

[30] J.P. Courtiat, C.A.S. Santos, C. Lohr, and B. Outtaj. Experience with rt-lotos, a temporal extension of the lotos formal description technique. Invited Paper in Computer Communications, January 2000.

[31] J-M. Courvreur and D. Poitrenaud. *Vérification et mise en œuvre des réseaux de Pétri*, chapter Dépliage pour la vérification de propriétés temporelles, pages 127–161. Hermès, 2003.

[32] K. Dahan and M. Laliberté. Réflexions autour de la notion d'interprétation de la musique électroacoustique. In *Actes des 13^{iemes} Journées d'Informatique Musicale (JIM 08), Albi, France*, Mars 2008.

[33] O. Delerue. *Spatialisation du son et programmation par contraintes : le système MusicSpace.* PhD thesis, Université Paris VI, Pierre et Marie Curie, 2004.

[34] M. Desainte-Catherine. *Informatique Musicale, du signal au signe musical*, chapter Modèles temporels pour la synthèse musicale. Hèrmes, 2004.

[35] M. Desainte-Catherine and A. Allombert. Interactive scores : A model for specifying temporal relations between interactive and static events. *Journal of New Music Research*, 34(4), December 2005.

[36] M. Desainte-Catherine and S. Marchand. Structured additive synthesis : Towards a model of sound timbre and electroacoustic music forms. In *Proc. of the ICMC-99 (International Computer Music Conference), Pekin (China)*, October 1999.

[37] M. Desainte-Catherine and S. Marchand. Vers un modèle pour unifier musique et son dans une composition multi-échelle. In *Actes des 6^{e} Journées d'informatique Musicale (JIM99), Paris*, pages 59–68, 1999.

[38] M. Diaz and P. Sénac. Time stream petri nets : A model for multimedia streams synchronization. In *Proc. of the first International Conference on Mutli-media Modelling (1993), Singapore*, November 1993.

[39] M. Diaz and P. Sénac. Time stream petri nets : A model for timed multimedia information. In R. Valette, editor, *Proc. of the 15th International Coference on Application and Theory of Petri nets, Saragosse, Spain*, pages 219–238. Springer Verlag, 1994.

[40] M. Diaz, P. Sénac, and P. De Saqui Sannes. Un modèle formel pour la spécification de la synchronisation multimédia en environnement distribué. In G. Bochmann, editor, *Proceedings of CFIP'93, Montréal, Canada.* Hermés Science Publications, 1993.

[41] E. W. Dijkstra. *EWD316 - A Short Introduction to the Art of Programming.* 1971.

[42] M. Diaz (dir). *Les réseaux de Pétri, Modèles fondamentaux.* Hermes, 2001.

[43] M. Diaz (dir). *Vérification et mise en œuvre des réseaux de Pétri.* Hermes, 2003.

[44] F. Dorian. *The History of Music in Performance : The Art of Musical Interpretation until Renaissance to our Day.* Norton and co., 1942.

[45] R. J. Duffin. Topology of series-parallel networks. *Journal of Mathematical Analysis and Applications*, 10 :303–313, 1965.

[46] H. Dufourt. *Musique, Pouvoir, Ecriture.* Christian Bourgeois, 1991.

[47] I. Durand and S. Schwer. A tool for reasoning about qualitative temporal information : the theory of s-languages with a lisp implementation. *Journal of Universal Computer Science*, 14(3), 2008.

[48] S. E. Elmaghraby. *Activity networks : project planning and control by network models.* John Wiley and sons, New York, 1977.

[49] J. Ferber. *Les systèmes multi-agents, Vers une intelligence collective.* InterEditions, 1995.

[50] P. Hanappe. *Design and Implementation of an Integrated Environnement for Music Composition and Synthesis.* PhD thesis, Université Paris 6/Ircam, 1999.

[51] W. Harvey, P. Stuckey, and A. Borning. Compiling constraint solving using projection. In *Proc. of the 1997 Conference on Principes and Practice of Constraint Programming (CP97)*, 1997.

[52] J. Haury. La grammaire de l'exécution musicale au clavier et le mouvement des touches. *Analyse Musicale*, 7, 1987.

[53] J. Haury. Développement d'une instrument d'interprétation de la musique pour personnes en situtation de déficience motrice. Technical report, APF, 2006.

[54] J. Haury. Un répertoire pour le clavier de deux touches - théorie, notation et application musicale. *Documents Numériques*, 11(3-4) :127–148, 2008.

[55] G. Haus and A. Rodriguez. Music description and processing by petri nets. *Advances on Petri Nets, Lecture Notes in Computer Science*, 340 :175–199, 1988.

[56] G. Haus and A. Rodriguez. Formal music representation ; a case study : the model of ravel's bolero by petri nets. *Music Processing, G. Haus Editor, Computer Music and Digital Audio Series*, pages 165–232, 1993.

[57] G. Haus and A. Sametti. Scoresynth : a system for the synthesis of music scores based on petri nets and a music algebra. *IEEE Journal*, 24(7) :56–60, July 1991.

[58] T. Holmes. *Electronic and Experimental Music : Pioneers in Technology and Composition.* Routledge, 2003.

[59] HyTime. *HyTime. Information Technology - Hypermedia/Time-based Structuring Language (HyTime). ISO/IEC DIS 10744, 8*, 1992.

[60] M. Jantzen and R. Valk. Formal properties of place/transition nets. In *Lecture Notes in Computer Science : Net Theory and Applications, Proc. of the Advanced Course on General Net Theory of Processes and Systems, Hamburg, 1979*, volume 84, pages 165–212. Springer Verlag, 1980.

[61] K. Jensen. *Colored Petri Nets, Basic Concepts, Analysis Methods and Practical Applications.* Springer, 1997.

[62] R. APT Krzysztof. *Principles of Constraints Programming.* Cambridge, 2003.

[63] Nelson Posse Lago and Fabio Kon. The quest for low latency. In *Proc. of the International Computer Music Conference (ICMC2004), Miami, USA*, pages 33–36, 2004.

[64] M. Laurson. Pwconstraints. In *X Colloquio di Informatica Musicale (Milano)*, pages 332–335, 1993.

[65] M. Laurson and J. Duthen. Patchwork, a graphic language in preform. In *Proc. of the International Computer Music Conference (ICMC89), Ohio State University*, 1989.

[66] C. L. MacKenzie and D. VanEerd. Rythmic precision in piano scale : Motor psychophysics and motor programming. In *Proc. of the* 13^e *Symposium on Attention and Performance*, pages 375–408, 1990.

[67] P. Manoury. La note et le son : un carnet de bord. *Contre-champs*, 11 :151–64, 1990.

[68] K. Mariott and P. Stuckey. *Programming with Constraints : An Introducion*. MIT Press, 1998.

[69] M. Mathews. *The Technology of Computer Music*. MIT Press, 1969.

[70] P.M. Merlin. *A study of the recoverability of computing systems*. PhD thesis, University of California, Irvine, 1974.

[71] M. Mesnage. Sur la modélisation des partitions musicales. *Analyse Musicale*, 22 :31–44, 1991. Repris dans *Formalisme et Modèles Musicaux num 1, ed. Delatour*.

[72] M. Moalla, J. Pulou, and J. Sifakis. Synchronized petri nets : A model for the description of non-autonomous sytems. *Lecture Notes in Computer Science*, 64 :374–384, 1978.

[73] F.R. Moore. *Elements of Computer Music*. Prentice-Hall, 1990.

[74] Y.E. Lien N.D. Jones, L.H. Landweber. Complexity of some problems in petri nets. *Theorical Computer Science*, 4 :277–299, 1977.

[75] Mogens Nielsen, Gordon Plotkin, and Glynn Winskel. Petri nets, event structures and domains. *Lecture Notes in Computer Science*, 70 :266–284, 1979.

[76] R. Willrich P. Sénac, P. De Saqui Sannes. Hierarchical time stream petri nets : A model for hypermedia systems. In G. De Michaelis and M. Diaz, editors, *Proc. of the 16th International Conference on Applications and Theory on Petri Nets, Turino, Italy*, pages 451–470, 1995.

[77] F. Pachet and P. Roy. *Informatique Musicale, du signal au signe musical*, chapter Harmonisation musicale automatique et programmation par contraintes. Hermès, 2004.

[78] C. Palmer. On the assignment of structure in music performance. *Music Perception*, 1996.

[79] C. A. Petri. *Kommunication mit Automaten*. PhD thesis, Bonn : Institut fur Instrumentelle Mathematik, 1962. English translation : Technical Report R5AC-TR-65-377, vol. 1, suppl. 1, Griffiths air force base, New-York.

[80] S. Pope. Music notations and the representation of musical structure and knowledge. *Perspectives of New Music*, 24(2), 1986.

[81] M. Puckette. Combining event and signal processing in the max graphical programming environment. *Computer Music Journal*, 15(3), 1991.

[82] M. Puckette. Pure data : another integrated computer music environment. In *Proc. of the Second Intercollege Computer Music Concerts, Tachikawa, Japan*, 1996.

[83] M. Puckette. Using pd as a score language. In *Pr. of the International Computer Music Conference (ICMC02), Goteborg, Sweden*, 2002.

[84] C. Ramchandani. *Analysis of asynchronous concurrent systems by timed Petri nets*. PhD thesis, MIT, Departement of electrical engineering, 1974.

[85] C. Rhodes and R. Strandh. Gsharp, un éditeur de partitions de musique interactif et personnalisable. *Documents Numériques*, 11(3-4) :9–28, 2008.

[86] M. Robine. *Analyse de la performance musicale et synthèse sonore rapide*. PhD thesis, Ecole Doctorale de Mathématiques et d'Informatique de l'Université de Bordeaux 1, 2006.

[87] C. Rosen. *La génération romantique*. Gallimard, 2002. Traduit de l'américain par G. Bloch.

[88] Pierre Roy. *Satisfaction de contraintes et programmation par objets*. PhD thesis, Université Paris VI, 1998.

[89] D. Rubine and P. McAvinney. Programmable finger-tracking instrument controllers. *Computer Music Journal*, 14(1) :26–40, 1990.

[90] Christian Schulte and Peter J. Stuckey. Efficient constraint propagation engines. *Transactions on Programming Languages and Systems*, 31(1) :2 :1–2 :43, December 2008.

[91] SMIL. *SMIL 3.0 W3C Recommendation*, December 2008.

[92] C. Truchet. *Contraintes, Recherche locale et Composition Assistée par Ordinateur*. PhD thesis, Université Paris VII, 2004.

[93] R. Valk. Self-modifying nets : A natural extension of petri nets. In *Proc. of the* 5^{th} *international conference on automates, languages and programming, ICALP'78,Udine, Italy, in Lecture Notes in Computer Science*, volume 62, pages 464–476, 1978.

[94] J. Vautard and A. Lallouet. Auralisation of a constraint solver. In *Proc. of Internation Compouter Music Conference (ICMC 2006), New Orleans (USA)*, pages 564–571, November 2006.

[95] M. Vilain. A system for reasoning about time. In *Proc. of the AAAI*, pages 197–201, 1982.

[96] A. M. Wing. Perturbations of auditary feedback delay and the timing of movement. *Journal of Experiment Psychology : Human Perc. and Performance*, 3(2) :175–186, 1977.

[97] I. Xenakis. *Musiques Formelles*. Stock Musique, 1981.

[98] W.M. Zuberek. Timed petri net and prelimenary performance evaluation. In *Proc. of the 7th Annual Symposium on Computer Architecture, La Baule, France*, 1980.