

Suivi de partitions musicales

rapport d'activité

—

IRCAM - Centre Pompidou
1, place Igor Stravinsky, 75004 Paris

Gilles Mathieu
Elève ingénieur, département Informatique,
INSA de Lyon
Gilles.Mathieu@ircam.fr - gilles.mathieu@insa-lyon.fr

Stage du 5 mai au 23 août 2002

Résumé

Le suivi de partitions musicales est un système de plus en plus utilisé par les compositeurs : il permet en effet d'améliorer la qualité des performances en considérant le dispositif informatique comme un accompagnateur et non plus comme un guide pour l'instrumentiste.

Le travail réalisé au cours de mon stage sur le suivi de partitions s'inscrit dans la continuité d'un ensemble de recherches menées au sein de l'équipe temps-réel de l'IRCAM, notamment par Nicola Orio. Ces recherches avaient mené à l'élaboration d'un système de suivi basé sur les modèles de Markov cachés.

Le présent rapport est à destination interne. Son but est de présenter le travail réalisé au cours de mon stage. Les conditions, buts et objectifs du stage sont décrits chapitre 1 ; La description des objets utilisateurs au chapitre 2 ; les chapitres 3 et 4 présentent une documentation interne de programmation ; enfin, ouvertures et autres considérations sont décrites au chapitre 5.

Parallèlement à ce rapport, plusieurs fichiers de documentation ont été réalisés afin de rendre mon travail utilisable par les personnes continuant à travailler sur le projet. Certains de ces fichiers sont joints en annexes.

Table des matières

1	Point de départ - Objectifs	5
1.1	Mon stage	5
1.2	Le système existant	5
1.2.1	Principe	5
1.2.2	Audio et MIDI	6
1.3	Les objectifs	6
1.3.1	Premiers objectifs	6
1.3.2	Objectifs "revus"	6
2	L'interface - Les objets jMax	7
2.1	Suivimidi : les changements	7
2.1.1	Paramètres d'entrée	7
2.1.2	Affichage en déboguage	7
2.1.3	Avance et retour forcé	8
2.2	Les nouveaux objets	8
2.2.1	Suivieval	8
2.2.2	Suivimakecue	9
2.2.3	Suivimapper	9
3	La modélisation	11
3.1	Unification suiviaudio / suivimidi	11
3.1.1	Séparation du score parsing	11
3.1.2	N-states et G-states pour suiviaudio	11
3.2	Traitement des trilles	11
3.3	Modélisation des états	12
3.3.1	Un nouveau type d'état : le ghost-rest	12
3.3.2	Nouvel ordre dans la modélisation	12
3.4	Découpage polyphonique	12
3.4.1	Limites de l'ancien modèle	12
3.4.2	La notion de "real event"	13
3.4.3	Les transitions entre notes	14
3.4.4	Difficultés - limites de l'approche	18
3.4.5	Solution retenue	18

4	Le suivi	21
4.1	Suivimidi : réécriture des algorithmes	21
4.1.1	Notes on - notes off	21
4.1.2	Mode nozigzag	21
4.1.3	Fonctions newnote et update	22
4.2	Le calcul des observations	23
4.2.1	Prise en compte de la polyphonie	23
4.2.2	Poids des attaques	23
5	Travail parallèle - travail en cours	25
5.1	Autres travaux sur le suivi	25
5.2	Développement futur	25
5.2.1	Restructuration	25
5.2.2	L'apprentissage	25
5.2.3	Le suivi "de contour"	26
5.2.4	Le portage vers jMax 3.1	26
5.2.5	Idées - souhaits	26
5.3	Conclusion - bilan du stage	27
A	Compte-rendus de réunions	29
A.1	Réunion du 7 mai	29
A.2	Réunion du 15 mai	30
A.3	Réunion du 12 juin	30
B	Historique du travail réalisé	33
C	Screenshots	39
	Bibliographie	47

Table des figures

3.1	Résultat de l'ancienne modélisation	13
3.2	Résultat de la nouvelle modélisation	13
3.3	Exemple de ONE_ATTACK et ONE_RELEASE	14
3.4	Exemple de POLY_ATTACK et POLY_RELEASE	14
3.5	Exemple de LEGATO	15
3.6	Exemple de POLY_LEGATO	15
C.1	Patch d'aide de l'objet suiviaudio	40
C.2	Patch d'aide de l'objet suivimidi	41
C.3	Patch d'aide de l'objet suivimakecue	42
C.4	Patch d'aide de l'objet suivimapper	43
C.5	Patch d'aide de l'objet suivieval	44
C.6	suvieval utilisé avec suivimidi	45
C.7	suvieval utilisé avec suiviaudio	46

Chapitre 1

Point de départ - Objectifs

1.1 Mon stage

Elève ingénieur au département Informatique de l'INSA de Lyon, ce stage se place pour moi dans le cadre d'un stage de formation de fin de 4ème année (2ème année de spécialisation).

Effectué du 5 mai au 23 août 2002, il portait sur le sujet suivant : *Développement d'un système de suivi de partitions musicales basé sur les Modèles de Markov Cachés.*

Equipe d'accueil : Equipe Systèmes Temps-réel.

Responsable de stage : François Déchelle.

Encadrement : François Déchelle, Diemo Schwarz.

Collaborateurs : Vincent Goudard, Nicola Orio, Diemo Schwarz

Soutien, Aide jMax et programmation : Riccardo Borghesi, François Déchelle, Norbert Schnell, Patrice Tisserand.

1.2 Le système existant

Le système à améliorer a été développé par Nicola Orio sur deux plates-formes :

- Sous la forme d'un package jMax, avec un ensemble de fonctions écrites en C et des objets de suivi utilisables en collaboration avec d'autres objets jMax ;
- Sous une version 'recherche' pour MATLAB, permettant une plus grande clarté de contrôle des algorithmes.

1.2.1 Principe

Le principe du système est l'utilisation des Modèles de Markov Cachés (HMMs) pour modéliser la partition et ensuite trouver la correspondance entre signal reçu et partition attendue (*voir à ce sujet les articles de Nicola Orio, François Déchelle et Diemo Schwarz, [OD01] et [OS01]*).

Le temps est également pris en compte dans ce modèle, voir [Mou01].

L'objet suiveur envoie en sortie la note de la partition reconnue comme étant la bonne, ainsi qu'un *label*, nombre entier préalablement associé à cette note par l'utilisateur, et utilisé comme déclenchement d'événements musicaux.

1.2.2 Audio et MIDI

Deux objets jMax ont été créés pour le suivi :

- **suiviaudio**, qui réalise le suivi à partir d'une performance d'entrée sous forme de signal audio. Le calcul des observations s'effectue sur un ensemble de paramètres d'analyse du signal (fréquence, bandes harmoniques, spectre, énergie...).

- **suivimidi**, qui prend en entrée un signal standard MIDI et réalise les observations à partir du pitch (hauteur) de la note.

Chaque objet a son propre modèle, ses propres algorithmes et son fonctionnement. Chacun répond à une attente particulière dans le domaine du suivi de partition.

1.3 Les objectifs

1.3.1 Premiers objectifs

Les premiers objectifs, définis avec François Déchelle, sont les suivants :

- **amélioration du système de suivi pour la voix chantée** : Concerne le suiviaudio. Le système existant a quelques problèmes avec la voix notamment sur les notes répétées ou les imprécisions dues aux vibratos ou ports de voix.

- **réflexion sur une reconnaissance de contours** : également pour suiviaudio. Certaines séquences peuvent être jouées 'approximativement justes' mais poser toutefois des problèmes au suiveur. L'idée est d'avoir une approche relative de la partition, qui permettrait de reconnaître une séquence qui aurait été légèrement transposé, ainsi qu'une séquence de notes rapides imprécises.

- **gestion de l'apprentissage** : rendre le modèle évolutif (ce que permet l'utilisation des HMMs).

- **débogage du suivimidi**, qui présente quelques bugs le rendant inutilisable.

1.3.2 Objectifs "revus"

Au cours de ma période de stage, en fonction des différentes attentes vis-à-vis du suivi et des nouveaux problèmes soulevés, les objectifs initiaux ont été revus. Il s'est ajouté :

- **Application du modèle au suivi polyphonique**, notamment pour l'utilisation avec le piano.

- **unification de suiviaudio et suivimidi**, dont seul le calcul des observations devrait être différent.

Les objectifs plus liés au suiviaudio sont peu à peu sortis de mon champ d'action pour entrer dans les objectifs principaux du stage de Vincent Goudard.

Chapitre 2

L'interface - Les objets jMax

2.1 Suivimidi : les changements

2.1.1 Paramètres d'entrée

Notes en entrée : L'ancienne version de suivimidi recevait en entrée une valeur entière correspondant au pitch de la note jouée. La nouvelle version prend une liste de deux éléments : *{pitch, vitesse}*.

Prendre en compte la vitesse permet ainsi de recevoir tant les notes-on que les notes-off, en les distinguant (une note-off étant une note MIDI de vitesse 0). Cette prise en compte des notes-off est un des axes d'amélioration du système de suivi (voir chapitre 4 : le suivi).

Cette modification nécessite une adaptation des patches jMax utilisant le suivi, comme décrit dans le fichier de démonstration *midi_demo.jmax*, ainsi que dans le fichier d'aide de l'objet suivimidi, accessible comme tous les fichiers d'aide des objets jMax.

Nouveaux modes et valeurs paramétrables : L'ajout d'un nouveau mode de fonctionnement (mode *nozigzag*) nécessite la définition de deux nouveaux messages en entrée de suivimidi : **zigzag** et **nozigzag**. Ces messages fonctionnent de la même façon que les modes préexistants **explemu** et **noexplemu**.

- La valeur **max_diff** devient un paramètre d'entrée de suivimidi. cette variable, dont la valeur est en millisecondes, représente l'écart de temps maximum entre deux notes pour que celles-ci soient considérées comme un accord. Sa valeur par défaut est de 30ms.

- La valeur **attack_weight** est également un paramètre. Elle peut être changée par le message "**attack_weight n**". Sa valeur par défaut est 0. Voir le chapitre 4 (le suivi) pour des précisions sur l'utilité de cette variable.

2.1.2 Affichage en débogage

Pour une visualisation plus efficace des résultats du suivi, un certain nombre de données peuvent être affichées à la demande de l'utilisateur : ces données sont regroupées de la manière suivante :

- Données reçues (notes-on et notes-off, avec le pitch correspondant) ;
- Notes reconnues (nouvelle note détectée, avec le pitch correspondant) ;
- Nombre de g-states traversés depuis le dernier n-state ;
- Avertissement lors d'un retour en arrière du suiveur ;

- Résultats statistiques une fois le suivi terminé.

Chacun de ces types d’affichage peut être obtenu en envoyant le message **verbose 0, 1**, etc jusqu’à 4 pour le dernier groupe. L’arrêt de l’affichage est obtenu par l’envoi du message **noverbose 0,1,...**

2.1.3 Avance et retour forcé

Deux nouveaux symboles ont été ajoutés pour le “pilotage” de suivimidi : “>>” (forward) et “<<” (backward). Ces messages envoyés à l’objet forcent le suiveur à reconnaître comme événement courant l’événement suivant de celui que les observations lui ont permis de détecter.

Il est possible d’effectuer ces bonds en avant ou en arrière avec un plus grand nombre de pas. Il suffit de donner à suivimidi ce nombre de pas dans le message (le message “>> 5” fera avancer le suiveur de 5 événements).

2.2 Les nouveaux objets

La bibliothèque d’objets pour le suivi de partition (package jMax **suivi**) s’est enrichie de trois objets *utilitaires*, destinés aux statistiques ou à l’amélioration des capacités du suivi. Ce sont les objets **suivieval**, **suivimakecue** et **suivimapper**.

A chacun de ces objets correspond un fichier d’aide accessible de la même façon que tout fichier d’aide d’objet jMax (voir annexes C).

2.2.1 Suivieval

L’objet **suivieval** est un objet statistique, utile en environnement de test, lorsque la performance est simulée par une séquence jMax jouée avec l’objet **seqplay**, ou par un fichier son. Il permet d’évaluer quantitativement la qualité du suivi, en comparant les temps à la milliseconde près, ce qui est impossible à faire visuellement simplement en regardant les séquences.

L’objet permet ainsi de comparer différents suiveurs tournant en parallèle, et d’avoir des résultats fiables liées aux conditions : version du suivi, mode employé...

A la séquence représentant la performance est associée une piste de labels, joués en même temps que les notes correspondantes. Cette piste servira de référence.

L’objet reçoit en entrée :

- Les notes jouées de la performance ;
- Les labels associés aux notes jouées ;
- Les notes reconnues par le suiveur ;
- Les labels sortis par le suiveur.

Des statistiques sont ensuite établies à partir de ces données : le temps de latence entre le moment où une note est jouée et le moment où elle est reconnue donne le nombre d’erreurs, le pourcentage de notes reconnues, l’écart-type,...

Les données sont sorties de deux manières : en temps réel via les outlets de l’objet, ainsi que dans quatre fichiers de log. Ces données sont :

- Le pourcentage de reconnaissance de toutes les notes jouées ;

- Le pourcentage de reconnaissance *dans les temps* des notes jouées ;
- Le nombre total d'erreurs (notes non reconnues + notes reconnues trop tard) ;
- Le nombre de notes non reconnues ;
- L'écart-type des temps de latence.

Un temps de latence maximum, **errorthreshold**, est paramétrable par l'utilisateur. Si une note est reconnue plus de *errorthreshold* ms après avoir été jouée, elle est considérée comme une erreur.

les modes du suivi (explemu, zigzag) sont récupérées par suivieval pour garder un contexte dans les fichiers de log.

N.B : Pour utiliser suivieval avec suiviaudio, il est nécessaire d'avoir une piste MIDI de référence correspondant à la performance audio pour pouvoir réaliser une comparaison entre ce qui est joué et ce qui est reconnu.

2.2.2 Suivimakecue

suivimakecue est un utilitaire dont la fonction est d'ajouter automatiquement une piste de labels à une séquence de notes. L'avantage est double :

- **Rapidité** : Il n'est pas nécessaire de jouer la séquence pour que les labels soit associés aux notes. Suivimakecue ne fonctionne pas en temps réel, mais récupère toutes les notes de la séquence pour faire l'association.

- **Efficacité** : Pour les accords notamment, puisqu'un seul label est associé à chaque accord, ce qui évite la multiplication des labels inutiles ou la tâche fastidieuse de supprimer les doublons à la main.

L'objet récupère en entrée une séquence jMAX de deux *tracks* : une *track* de notes (la séquence à indexer) et une *track* d'entiers (la piste où vont être ajoutés les labels). La mise en route se fait par l'envoi d'un message de type *bang*. L'intervalle **max_diff**, définissant le temps maximum entre deux attaques d'un même accord, est paramétrable.

Par défaut, suivimakecue indexe uniquement les attaques de notes. Mais il est également possible d'ajouter des labels à tous les événements (notes et silences).

2.2.3 Suivimapper

L'objet suivimakecue présente néanmoins un inconvénient : il associe un label à chaque événement musical (note ou accord), alors que seule une partie de ces événements sert au déclenchement.

L'idée de suivimapper est donc la suivante : récupérer les labels sortis par le suiveur, correspondant plus ou moins à un label par note, et associer à ce label d'entrée (que nous appellerons *label-note*) un label de sortie si l'événement est déclencheur, et ne rien sortir du tout dans le cas contraire.

Ceci a également pour effet d'améliorer la robustesse du suiveur : en effet, on peut associer à un label de déclenchement un intervalle de plusieurs labels-notes ; le premier label-note reçu de cet intervalle provoquera l'envoi du label de déclenchement associé.

Prenons un exemple. Supposons une partition de 10 notes. A chaque note est associé un label, numéroté de 1 à 10. Supposons également que trois événements doivent être déclenchés par le suivi de cette partition : le premier sur la note 3, le deuxième entre les notes 5 et 7, et le dernier sur la note 9. Nous allons donc donner à l'objet `suivimapper` les listes suivantes en entrée :

`{1,3,3}`, `{2,5,7}` et `{3,9,9}`

ainsi, la réception du label 3 déclenchera l'envoi du label 1, le premier label reçu parmi (5,6,7) déclenchera l'envoi du label 2, et la réception du label 9 déclenchera l'envoi du label 3.

Rien ne se passe lorsqu'on reçoit les labels 1, 2, 4, 8 et 10. De plus, chaque label n'est sorti qu'une fois, ce qui évite les éventuels problèmes de double déclenchement en cas de retour en arrière du suiveur.

D'autre part, le fait de pouvoir associer plusieurs labels-notes à un label de déclenchement permet de diminuer la probabilité de sauter un label dans le cas où celui-ci se trouve sur une note rapide. Il suffit pour cela d'associer au label de déclenchement les quelques notes entourant celle sensée déclencher l'événement. La première de ces notes qui sera reconnue provoquera le déclenchement.

Enfin, les paramètres d'entrée étant des listes, il est possible de donner à l'objet les valeurs de référence en utilisant l'objet *pbank* de `jmax`. Les listes d'associations entre labels peuvent ainsi être stockées dans un fichier texte, ce qui est plus pratique et ergonomique que d'envoyer ces listes à `suivimapper` une par une via un patch `jMax`.

Chapitre 3

La modélisation

3.1 Unification suiviaudio / suivimidi

3.1.1 Séparation du score parsing

Dans la version de départ du système de suivi, le suivi destiné à l’audio et celui destiné au MIDI étaient distincts à toutes les étapes : modélisation et suivi. Si effectivement le mécanisme de suivi ne peut pas être le même, en revanche la modélisation de la partition devrait être indépendante du type de suivi.

L’objectif a donc été le suivant : Séparer, pour suivimidi, la partie modélisation de la partie suivi, et ensuite utiliser cette partie modélisation pour suiviaudio. Le **score parsing** se trouve donc désormais dans le fichier **suivimakenet.c**. C’est là que se trouvent les fonctions **makenet** et **refill** appelées par suivimidi et suiviaudio, et qui récupèrent la séquence servant de partition pour créer le modèle.

3.1.2 N-states et G-states pour suiviaudio

Le modèle décrit dans [OD01] par Nicola Orio et François Déchelle n’était pas réellement implémenté dans suiviaudio, alors qu’il l’était plus ou moins dans suivimidi.

L’unification des *score parsing* pour l’audio et le MIDI a permis d’avoir ainsi le même modèle : une suite de *normal states* représentant une entité musicale (note ou accord), avec son *ghost state* associé.

3.2 Traitement des trilles

Dans l’état actuel d’avancement du travail, il n’y a pas de traitement particulier pour les trilles : ceux-ci sont codés comme des accords dans la partition et traités comme tels.

Néanmoins, et en vue de la mise en place future d’un traitement spécial, la représentation d’un trille a changé :

Avec l’ancien modèle, un trille était codé au moyen de la vitesse : une note avec une vitesse inférieure à 10 indiquait que cette note était un trille, et que la deuxième note du trille était représentée par cette vitesse (*une note de la partition notée pitch 63 et vitesse 2 indiquait un trille entre la note 63 et la note $63+2 = 65$*).

Cette manière de coder les choses n'est pas très robuste car il est possible d'avoir des notes à très faible vélocité, qui risquent du coup d'être interprétées comme des trilles.

Pour la nouvelle modélisation, on considère comme trille un accord codé sur un canal MIDI différent. Cela ne change rien pour l'instant au niveau du suivi, mais permet de détecter à quel endroit un trille est attendu, ce qui peut être utile si on décide de faire quelque chose de particulier pour les trilles...

3.3 Modélisation des états

3.3.1 Un nouveau type d'état : le ghost-rest

L'une des suppositions de base de l'ancien modèle était la suivante : Il est impossible de jouer faux un silence. Cette supposition n'est pas stupide *a priori*, mais n'est hélas pas toujours vérifiée : il est en effet possible de *sauter* un silence. Cela implique qu'il faut prévoir dans la modélisation que l'on puisse également ne pas passer dans l'état de type **rest** correspondant au silence.

D'où l'utilité d'un nouveau type d'état : le **G-REST**.

La modélisation d'un silence se rapproche donc de plus en plus de celle d'une note : si la note est correctement jouée, nous aurons une forte probabilité d'être dans le **N-STATE** de cette note. Si elle n'est pas bien jouée, la probabilité sera plus forte d'être dans le **G-STATE**.

De même, pour les silences, si aucune note n'est jouée la probabilité est maximale dans le **REST**. Si une note est jouée nous pourrions passer outre le **REST** via l'état correspondant **G-REST**.

3.3.2 Nouvel ordre dans la modélisation

Le modèle présenté "à plat" tel qu'il est décrit dans [OD01] ne peut pas être réellement implémenté tel quel. Il n'est pas concevable d'imaginer deux chaînes parallèles, l'une de **N-STATES** et l'autre de **G-STATES**. Les états sont donc créés sous forme d'une chaîne, et l'initialisation des transitions entre ces états donnera la topographie voulue.

Afin d'améliorer et de simplifier l'initialisation de ces transitions, l'ordre de création et de chaînage des états a été modifié. Le modèle est toujours une succession de **N-STATES** et de **G-STATES** (ou de **RESTS** et de **G-RESTS**), mais contrairement à l'ancien modèle, **les ghost-states sont modélisés avant leurs normal-states associés**. Ceci permet ainsi une meilleure correspondance entre le modèle théorique et le modèle implémenté, dans la mesure où les transitions s'effectuent toujours vers l'avant entre les états de la chaîne.

3.4 Découpage polyphonique

3.4.1 Limites de l'ancien modèle

Modéliser un état par note ou par accord ne pose pas de problème dans une partition monophonique, ou une partition polyphonique dont les seuls événements sont des accords.

Malheureusement, ceci ne fonctionne pas pour les situations où les événements sont polyphoniques sans être des accords : notes tenues en pédale, recouvrement de notes...

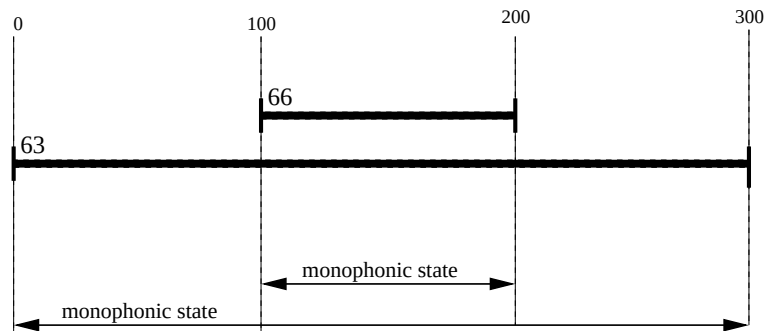


FIG. 3.1 – Résultat de l'ancienne modélisation

Dans la figure 3.1 par exemple, avec un état par note, nous aurions un état pour la note de pitch 63 avec une durée de 300, et un état pour la note 66 avec une durée de 100. Evidemment, cela ne correspond à rien et pose donc un problème au suivi.

3.4.2 La notion de "real event"

L'idée est donc de considérer comme événement non plus chaque note, mais chaque changement dans la partition, qu'il soit début ou fin de note. On appellera ces changements "real events".

Une structure **realevent_t** a été implémentée pour répondre à cela. On pourra ainsi considérer qu'un état correspond à l'intervalle entre deux real events. Pour l'exemple précédent, nous aurons donc quatre real events :

- Le début de la note 63, au temps 0 ;
- Le début de la note 66, au temps 100 ;
- La fin de la note 63, au temps 200 ;
- La fin de la note 66, au temps 300.

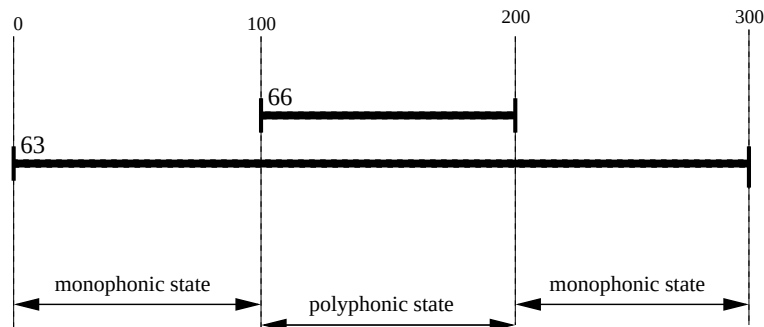


FIG. 3.2 – Résultat de la nouvelle modélisation

la modélisation associée sera donc suivant la figure 3.2. Cette succession d'états correspond beaucoup mieux à la partition que la version précédente !

3.4.3 Les transitions entre notes

La limite de cette approche apparaît assez vite : il n'est pas forcément nécessaire de modéliser systématiquement un état entre chaque real event. C'est le cas typique d'un accord dont toutes les notes ne sont pas attaquées exactement en même temps : il faut un état pour l'accord, mais pas un état pour chaque intervalle entre deux attaques.

Pour faire face à cela, il convient de regrouper en un seul et même événement tous les événements survenant dans un intervalle minimum appelé **max_diff**. Le regroupement et la modélisation des états va donc suivre différents cas de figure :

ONE_ATTACK et **ONE_RELEASE** : une seule note est attaquée ou relâchée dans un intervalle de $\pm$ **max_diff** (figure 3.3) ;

POLY_ATTACK et **POLY_RELEASE** : Plusieurs notes sont attaquées dans l'intervalle, ou plusieurs notes sont relâchées dans cette intervalle (figure 3.4) ;

LEGATO : Une note se termine et une autre commence à l'intérieur du même intervalle (avec ou sans recouvrement) (figure 3.5) ;

POLY_LEGATO : plusieurs notes se terminent et commencent dans le même intervalle (figure 3.6).

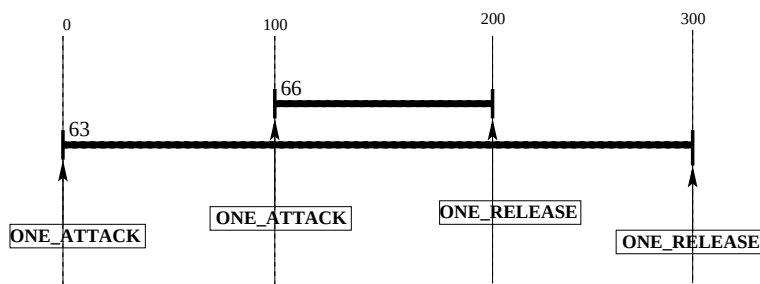


FIG. 3.3 – Exemple de ONE_ATTACK et ONE_RELEASE

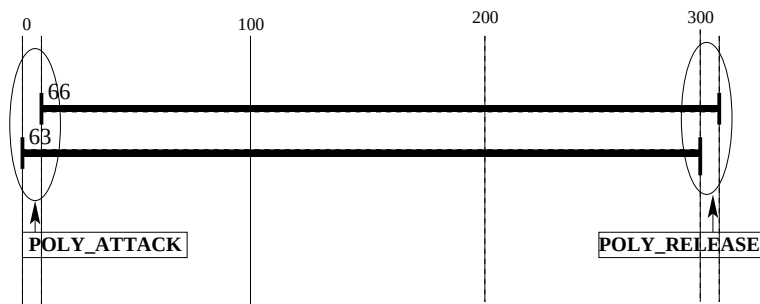


FIG. 3.4 – Exemple de POLY_ATTACK et POLY_RELEASE

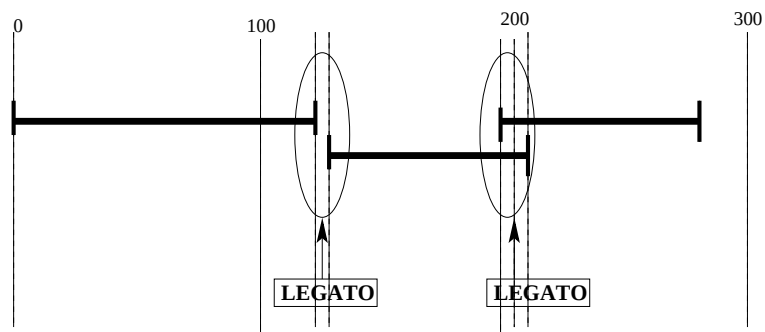


FIG. 3.5 – Exemple de LEGATO

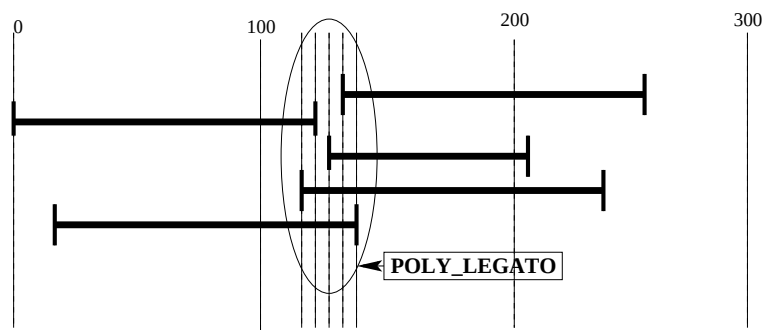


FIG. 3.6 – Exemple de POLY_LEGATO

Les différents cas de figures sont déterminés de la manière suivante :

DEBUT

```
dur = temps entre le real_event courant et le suivant

si ( dur > max_diff ), alors
    si ( le real_event est un debut ), alors
        type = ONE_ATTACK;
    sinon
        type = ONE_RELEASE;
    fin si
sinon
    si ( le real_event est un debut ), alors
        si ( le real_event suivant est un debut ), alors
            type = POLY_ATTACK;
        sinon
            type = LEGATO;
        fin si
    sinon
        si ( il y a un debut et une fin dans l'intervalle max_diff ), alors
            type = POLY_LEGATO;
        sinon
            si ( le real_event suivant est un debut ), alors
                type = POLY_RELEASE;
            sinon
                type = LEGATO;
            fin si
        fin si
    fin si
fin si
```

FIN

- Ceci permet de traiter tous les cas pouvant se présenter :
- Evenements simples (ONE_ATTACK, ONE_RELEASE),
 - événements multiples de même nature (POLY_ATTACK, POLY_RELEASE),
 - événements “doubles” de nature différente (LEGATO),
 - événements de nature différente en nombre indifférent (POLY_LEGATO) .

L'algorithme de traitement pour ces différents cas de figures est le suivant :

SELON (*le type de transition*) :

type ONE_ATTACK :

Modéliser un N-state et un G-state correspondant aux notes;

type ONE_RELEASE :

si (*il y a des notes tenues*), alors
 modéliser un N-state et un G-state;
sinon
 modéliser un silence;
fin si

type POLY_ATTACK :

 récupérer toutes les notes dans l'intervalle de max_diff;
 ajouter les notes à la séquence de notes de l'état;
Modéliser un N-state et un G-state;

type POLY_RELEASE :

 Récupérer la dernière fin de note;
 si (*il y a des notes tenues*), alors
 modéliser un N-state et un G-state;
 sinon
 modéliser un silence;
fin si

type LEGATO :

 si (*il y a un recouvrement*), alors
 modéliser un N-state et un G-state;
 Ignorer la fin de l'événement suivant;
 sinon
 ignorer la fin de la note ;
fin si
Passer à l'événement suivant;

type POLY_LEGATO :

 Ignorer les fins de notes;
 récupérer toutes les notes commençant dans l'intervalle de max_diff;
 ajouter les notes à la séquence de notes de l'état;
Modéliser un N-state et un G-state;

FIN SELON

3.4.4 Difficultés - limites de l'approche

La gestion des notes tenues

Les états à modéliser dépendent non seulement du type de “transitions” décrit précédemment, mais également de la présence ou non de notes tenues au moment de ces transitions.

En effet, une transition de type **ONE_RELEASE** par exemple peut donner lieu à deux modélisations suivant le contexte : un N-STATE et un G-STATE s'il y a d'autres notes tenues, ou un REST et un G-REST s'il n'y en a pas.

Il faut donc, à chaque **real_event**, associer d'une part le nombre de notes tenues à ce moment là, et d'autre part garder les références de ces notes pour pouvoir le cas échéant modéliser un état leur correspondant.

Pour cela, lorsqu'on récupère un événement, chaque note attaquée et non terminée à l'instant *t* est récupérée dans un tableau. Il suffit ensuite d'ajouter à la polyphonie de la note courante les notes du tableau, qui correspondent aux notes tenues.

Le problème lié à ce traitement est qu'il est réalisé dans la fonction **suivimakenet_addevent**, appelée par **suivimakenet_makenet** et **suivimakenet_refill**.

Cela signifie que les notes tenues ne sont pas gardées en mémoire d'un refill à l'autre. On peut donc se trouver avec des états modélisés ne correspondant pas à la réalité si d'aventure un refill est effectué alors qu'il y avait des notes tenues...

Le mécanisme mis en place est du même ordre que celui existant pour le refill des silences que l'on n'aurait pas eu la place d'ajouter :

À la fin de chaque refill, les notes tenues du dernier état modélisé sont stockées dans une variable de la structure *netlev* correspondant au modèle, et ainsi récupérées lors du refill suivant.

Le nombre d'événements à “découper”

Par ailleurs, découper la séquence à chaque refill pose un autre problème : ce découpage doit donc être effectué pour permettre de créer le nombre d'états souhaités.

Le problème vient du fait qu'il n'y a pas correspondance entre le nombre de **real_events** et le nombre d'états modélisés. Le nombre de **real_events** créés à partir des événements de la séquence peut donc s'avérer insuffisant.

Prenons pour exemple un cas assez pathologique, mais néanmoins illustrateur : une partition débutant par une série d'accords de 10 notes.

Lors de la réalisation du modèle, nous allons prendre les **MAX_EVNT** premiers événements (**MAX_EVNT** = 40), et effectuer le découpage en **real_events**. Compte-tenu des règles de regroupement, et en toute logique, un seul état va être modélisé par accord, ce qui signifie que 10 **real_events** de type “begin” vont donner lieu à la création d'un seul état.

Nous nous retrouvons donc avec seulement quatre états modélisés après avoir épuisé notre “stock” de 40 **real_events**...

3.4.5 Solution retenue

Pour faire face à ces problèmes, la solution la plus simple est de réaliser le découpage de la séquence entière dès la première modélisation. Ceci est certes coûteux en place mémoire, mais facilite grandement les algorithmes et rend la modélisation beaucoup plus fiable : on évite ainsi

tous les problèmes liés aux *old_alivenotes* et aux *old_rests*, ainsi que le risque de ne pas avoir assez de **real_events** pour modéliser nos états.

Une allocation mémoire minimum est effectuée à l'initialisation du suiveur. Puis on récupère le nombre d'événements de la séquence entière afin d'allouer la mémoire nécessaire.

La fonction **suivimakenet_makenet** appelle une fonction *suivimakenet_cutseq*, dans laquelle le découpage est réalisé dans son intégralité. A chaque refill, on récupère la position du dernier **real_event** pour pouvoir modéliser la suite.

La mémoire est libérée à chaque destruction de l'objet suiveur.

Chapitre 4

Le suivi

4.1 Suivimidi : réécriture des algorithmes

4.1.1 Notes on - notes off

La modélisation décrite précédemment suppose que l'on soit capable de reconnaître le début et la fin de chaque note. Il est donc nécessaire de recevoir les informations correspondantes de la performance, à savoir les notes-on et les notes-off du protocole MIDI.

Ceci est rendu possible par le nouveau type des paramètres d'entrée, décrit au chapitre 2, section 2.1.1. En effet, une note-off, qui symbolise la fin de la note, n'est ni plus ni moins qu'une note avec une vélocité nulle.

Chaque événement reçu est donc identifié comme étant un début ou une fin de note au moyen de sa vélocité.

Cette prise en compte des fins de notes, inexistante dans la version de départ, permet de reconnaître plus efficacement les changements d'états et les silence éventuels. Avec la nouvelle modélisation, où un changement d'état peut intervenir sur une fin de note, ce traitement est indispensable.

4.1.2 Mode nozigzag

Il n'est pas rare que le suiveur "jongle" entre plusieurs états. Ces hésitations se traduisent par des zig-zags entre les états candidats et plus ou moins équiprobables.

Ceci est somme toute assez normal, puisque à chaque tick d'alarme la meilleure position est recalculée. Par contre, ce comportement peut se révéler gênant si l'émission d'un label associé à l'état sur lequel on revient est effectué plusieurs fois.

Pour palier à cela, il est possible d'utiliser suivimidi en mode **nozigzag**, via les message-boxes de l'interface (cf chapitre 2).

La principale fonctionnalité de ce mode est de considérer uniquement comme bons les états qui suivent le dernier reconnu. Ainsi les labels notes sorti(e)s par le suiveur suivent forcément l'ordre chronologique, et sont du même coup sortis une seule fois.

N.B : L'utilisation de l'objet **suivimapper** en sortie de suivimidi (cf chapitre 2 - les nouveaux objets) rend plus ou moins obsolète le recours au mode nozigzag, puisque suivimapper interdit également de sortir plusieurs fois le même label

4.1.3 Fonctions newnote et update

La fonction **suivimidi_newnote** est appelée à chaque fois que suivimidi reçoit quelque chose de la performance.

C'est dans cette fonction qu'est mis à jour le paramètre d'attaque pour la nouvelle note reçue, paramètre qui sera utilisé au moment de la reconnaissance pour différencier une note tenue d'une note qui vient d'être attaquée.

Ce paramètre d'attaque sert également à différencier les débuts de notes (attack = 1) et les fin de notes (attack = -1).

La fonction **suivimidi_update** est appelée par **suivimidi_newnote**, ainsi que par la fonction d'alarme (par défaut tous les **INT_TICK** = 5ms).

L'algorithme général de cette fonction a été complètement revu. Le principe n'a pas changé mais l'ajout de nouvelles fonctionnalités ainsi que de nouveaux blocs conditionnels rendait le total trop illisible, et donc susceptible d'être source d'erreurs et d'inefficacité.

En particulier, ont été ajoutés : Les sorties "d'erreurs" pour les g-states et g-rests, ainsi que la possibilité d'associer un label à un REST.

Voici sa nouvelle forme :

Debut

```
Calcul des Observations (suivimidi_obscalc)
decodage des Observations (suivimidi_decoding)
```

SELON (*le type de l'événement le plus probable*)

type G_STATE et type G_REST

Envoyer l'index de l'état en sortie

type LAST

Envoyer -1 en sortie

type REST

Envoyer le temps courant en sortie

si (*il y a un label associé*), alors

envoyer le label en sortie

fin si

Enregistrer l'état comme dernier état correct reconnu

Pointer sur l'événement suivant

type N_STATE

Envoyer le temps courant en sortie

Envoyer l'événement reconnu en sortie

Signaler l'événement reconnu dans la séquence (highlight)

si (*il y a un label associé*), alors

envoyer le label en sortie

fin si

```

    Enregistrer l'état comme dernier état correct reconnu
    Pointer sur l'événement suivant
    si ( notes reconnues > moitié du buffer ), alors
        effectuer un refill
    fin si

```

FIN SELON

activer l'alarme pour le prochain update

Fin

4.2 Le calcul des observations

4.2.1 Prise en compte de la polyphonie

Le pitch de la note courante était jusqu'à présent contenu dans la structure *suivimidi_t* sous la forme d'une valeur entière.

Mais une seule valeur ne permet pas de prendre en compte la polyphonie et l'attaque.

Une nouvelle structure, *curnote_t*, a été définie pour cela par Vincent Goudard. Une variable de type *curnote_t* contient une vélocité et une attaque, et la structure *suivimidi_t* contient un tableau de 128 *curnote_t*, un par pitch de note.

Ainsi, à chaque instant, il est possible de savoir quelles notes sont jouées ou non, et quelles notes sont attaquées à l'instant précis.

Ceci permet une comparaison directe avec le modèle, où sont également différenciés les notes attaquées des notes tenues.

4.2.2 Poids des attaques

Le suivi polyphonique d'un instrument comme le piano soulève un autre problème : Les débuts de notes sont précis, mais les fins le sont beaucoup moins.

Voici un cas particulier assez illustratif : Une note notée comme étant une ronde dans la partition doit être tenue pendant l'exécution d'autres notes. A partir de la partition, le suiveur va donc modéliser une suite d'états polyphoniques avec toujours la note tenue en fond.

A l'exécution, l'interprète peut très bien décider de tenir cette note à la pédale. Il va donc attaquer la note et la relâcher très vite, avant d'exécuter les autres. La plupart des pianos MIDI ne prenant pas la pédale en compte, la note va être considérée comme étant jouée très courte, et n'apparaîtra plus lorsque les notes suivantes seront jouées.

Il est donc important de faire la considération suivante : une attaque est toujours plus significative qu'une note tenue pour déterminer les probabilités d'être ou non dans tel ou tel état.

D'où l'intérêt de donner un poids aux notes attaquées lors du calcul des observation. C'est à cela que va servir le flag d'attaque dans la structure *curnote*.

Ainsi, plutôt que de calculer une “moyenne” de notes justes, on peut calculer une moyenne pondérée par un paramètre **ATTACK_WEIGHT**, ce qui permettra d'avoir une forte probabilité d'être dans l'état i si on a reçu l'attaque que l'on attendait, même si les notes tenues qui doivent apparaître aussi ne sont pas reconnues.

Chapitre 5

Travail parallèle - travail en cours

5.1 Autres travaux sur le suivi

Une bonne part du travail sur le suivi de partitions a été menée en équipe et parallèlement à mon stage. Il s'agit :

- Des séances de tests réalisés avec Andrew Gerszo et la flûtiste Emmanuelle Ophèle-Gaubert sur la pièce de Pierre Boulez “Explosante Fixe” ;
- Des tests réalisés avec Serge Lemouton et Philippe Manoury sur les pièces de P. Manoury “Pluton” et “En Echo” (avec la chanteuse Donatienne Michel-Dansac) ;
- Des travaux effectués par Diemo Schwarz et Nicola Orio ;
- Du stage de Vincent Goudard, parallèlement au mien, sur le suivi de partitions.

Afin de coordonner le projet, une liste de diffusion a été créée parmi les listes de l'IRCAM. Il s'agit de la liste **score-recognition@listes.ircam.fr**

5.2 Développement futur

5.2.1 Restructuration

Une complète réorganisation des fichiers est en cours, dans le cadre de l'unification MIDI / Audio. L'idée est de différencier les parties totalement indépendantes du type de suivi (création du modèle, décodage) des parties liées au mode utilisé (calcul d'observations, probabilités...)

Cela suppose une restructuration du modèle de l'application, notamment dans la définition des structures : il convient de séparer les différents paramètres de manière à ce que chacun ne connaisse que ce qui lui est utile. Cette partie du travail sera poursuivie par Vincent Goudard et Diemo Schwarz.

5.2.2 L'apprentissage

Plusieurs pistes ont été évoquées pour aborder le problème de l'apprentissage. L'une des idées serait d'utiliser les données de l'objet suivieval afin de récupérer les temps “réels” de sortie des événements pour les comparer au modèle et effectuer une moyenne pondérée. Ceci permettrait d'intégrer au modèle les différences entre la partition codée et ce qui est joué le plus souvent, et ainsi de construire chaque fois un modèle plus proche de la réalité.

Ceci suppose une importante restructuration du score parsing et une réflexion sur la façon de garder et d'assimiler les données "appprises".

5.2.3 Le suivi "de contour"

Il pourrait être possible d'effectuer ce suivi de contour évoqué par Philippe Manoury de plusieurs façons.

On peut imaginer, par exemple, un paramètre qui définirait le taux de "flou" à accorder à certaines parties de la séquence, permettant de ratisser plus large dans les bandes de fréquence du filtre, et ainsi considérer comme justes des notes jouées à plus ou moins un intervalle donné, un ton par exemple.

On peut également définir un paramètre définissant la valeur de référence du La, par défaut 440Hz, valeur qui pourrait être réajustable en cours de suivi en fonction de l'évolution générale de la tonalité...

Ces changements concernent principalement le suiviaudio. Une adaptation pourrait être envisagée pour le MIDI, dans le cas par exemple d'un grand arpège au piano qu'il n'est pas nécessaire (ni souhaitable) de reconnaître note à note, mais plutôt dans son emsemble.

5.2.4 Le portage vers jMax 3.1

L'essentiel du développement ayant été mené sur la version 2.5.3, un portage du système de suivi devra être effectué vers la version 3.1.

A partir du moment où ce portage sera décidé et effectué, tous les développements relatifs au suivi devront se faire sur cette version. Il est donc nécessaire d'avoir une version du suivi relativement stable, afin de limiter le nombre de problèmes à régler...

5.2.5 Idées - souhaits

Les avis divergent encore sur une question : la place à accorder au suivi MIDI par rapport au suivi Audio.

Une première tendance serait d'abandonner progressivement le MIDI au profit de l'audio, plus fiable et moins réducteur. Les différents tests effectués avec flûte MIDI, comparés aux tests effectués sur le signal audio vont dans ce sens, dans la mesure où la "modification" du signal d'un instrument acoustique est relativement hasardeuse.

Dans l'autre sens, le MIDI semble encore indispensable dans certains cas, notamment pour le piano : un signal MIDI issu d'un piano est relativement fiable, et d'autre part la polyphonie du signal se pose comme un obstacle à la reconnaissance audio.

Une autre idée évoquée concerne la modélisation de la partition. Il pourrait être possible, par exemple, de créer le modèle à partir d'une performance audio, plutôt qu'à partir d'une partition MIDI.

Il s'agit en fait d'une version poussée du mécanisme d'apprentissage, dans la mesure où chaque nouvelle exécution doit pouvoir permettre d'améliorer le modèle.

Par ailleurs, la réalisation d'une interface graphique reste à l'ordre du jour. Il reste donc à définir quel va être le support pour les données...

La version actuelle de l'interface est une interface jMax. L'utilisation de l'objet **sequence** comme représentation de la partition a ses avantages, mais aussi ses limites, notamment dans

le cas de suivi polyphonique. Il n'est pas toujours évident de savoir exactement dans quel état du modèle se trouve le suiveur...

Si une nouvelle interface doit être réalisée, cela suppose tout d'abord une importante réflexion sur la représentation choisie.

5.3 Conclusion - bilan du stage

Travailler sur le suivi de partitions, c'est un peu comme chercher du pétrole. Le but final est, bien sûr, de trouver du pétrole et de l'exploiter. Mais pour y arriver, il faut se définir de nombreux buts intermédiaires, et résoudre un certain nombre de problèmes qui n'apparaissent pas forcément de prime abord : Trouver où creuser, réparer le matériel, étayer les forages, bétonner les poches...

Bref, tout ça pour dire qu'il est difficile de répondre à la question : *“Les objectifs ont-ils été atteints ?”*.

Les principales conclusions peuvent se résumer ainsi :

- Les performances du suivi MIDI ont été améliorées de manière assez significatives, notamment dans le cas du suivi polyphonique.
- D'autre part, le travail a énormément avancé en ce qui concerne l'unification des suivis MIDI et audio.
- Le modèle créé à partir de la partition correspond bien mieux à sa description théorique, et colle d'autant plus à la partition “réelle”.
- D'autres exigences liées au travail (tests, interface,...) ont donné naissance à de nouveaux objets jMax (cf. chapitre 2) qui facilitent l'utilisation du suivi et améliorent son fonctionnement.
- Enfin, la mise à plat de nombreux problèmes et la réflexion commune sur le sujet de plusieurs personnes pendant 4 mois aura, je pense, permis au projet de sortir de l'immobilité et de redevenir un thème de recherche active.

Alors, a-t-on trouvé du pétrole ?

Disons, pour simplifier, que le travail est allé dans le bon sens. Si tout ne fonctionne pas encore à merveille comme une machine bien huilée, les nombreux changements et les réflexions collectives ont réellement fait avancer le projet.

Il est difficile de chiffrer cet avancement. Le travail n'est pas terminé, il reste certainement beaucoup à faire sur le sujet. Mais ce qui a été réalisé durant les quatre mois de mon stage sera, je l'espère, très utile pour la suite des travaux. Nous n'avons jamais été aussi proches de la poche de pétrole...

Annexe A

Compte-rendus de réunions

A.1 Réunion du 7 mai

Présents à la réunion : Serge Lemouton, Philippe Manoury, Diemo Schwarz, Gilles Mathieu.

Thème de la réunion : Discussion sur les attentes des utilisateurs quant au système de suivi de partitions.

Résumé : La problématique générale du système est la suivante : le système informatique doit pouvoir jouer le rôle d'accompagnateur vis à vis de l'instrumentiste qui le commande.

Cela implique une certaine adaptivité du système aux variations de tempo, de notes ou autres de la part du performeur par rapport à la partition.

L'idée est évoquée d'une possibilité d'apprentissage, à l'instar d'un accompagnateur humain. Il pourrait être souhaitable que le modèle de reconnaissance s'enrichisse au fur et à mesure des performances pour intégrer les variations qui reviennent souvent et ainsi être plus à même d'y faire face.

Se pose ensuite le problème du travail à effectuer. On peut en extraire deux parties :

- Une partie purement algorithmique, portant sur l'amélioration et l'extension du système actuel (meilleure reconnaissance pour la voix, pour les doubles notes...)
- Une partie axée IHM, qui pose le problème de la représentation des données : comment doit être matérialisé le suivi (Partition suivie type MIDI, représentation graphique...)

Enfin est évoqué le problème du travail à partir d'une source MIDI.

Une idée d'amélioration serait de travailler sur un signal audio plutôt que sur un modèle MIDI pour établir le modèle de base. Ceci pourrait être utile pour permettre l'apprentissage. D'autre part, l'information contenue dans la source MIDI est jugée un peu trop restreinte.

Toutefois, il apparaît que les utilisateurs restent attachés au MIDI. Premièrement pour une question de représentation et de suivi pour l'utilisateur, deuxièmement car le MIDI s'avère plus utilisable dans certains cas, comme par exemple le cas de reconnaissance polyphonique, où il serait plus difficile de différencier plusieurs fréquences fondamentales correspondant aux différentes voix par une analyse spectrale.

Il est également question d'un système de reconnaissance qui travaillerait en relatif, et non plus en absolu. Il peut arriver en effet qu'une suite très rapide de notes ne puisse pas être

détectée correctement. Il suffirait alors de reconnaître la note de départ, la note d'arrivée, et le mouvement général de la séquence (montée, descente...).

D'autre part, il est aussi possible qu'une séquence entière soit juste relativement, mais commencée sur une note trop basse ou trop haute. Le décalage doit pouvoir être rattrapé, et le système ne doit pas reconnaître comme fausse toute la séquence.

Il s'agirait donc en quelque sorte d'une reconnaissance de "contours", utilisable surtout pour la voix dans le cas de récitatifs.

A.2 Réunion du 15 mai

Présents à la réunion : Andrew Gerzso, Diemo Schwarz, Gilles Mathieu.

Thème de la réunion : Discussion et démo sur les bugs du suivi MIDI.

Résumé : Andrew Gerzso travaille actuellement sur la pièce "Explosante Fixe" de Pierre Boulez, pièce pour flûte et dispositif temps réel. Il utilise actuellement le système de suivi "NEXT", mais souhaite utiliser le nouveau suivi pour un concert au mois d'août.

Le principe est le suivant : à certaines notes correspondent des signaux de déclenchement, utilisés pour lancer ou arrêter des processus musicaux. Le système fonctionne assez bien pour certains essais. Par contre, l'essai sur une séquence comprenant de nombreuses notes répétées et rapides provoque un grand décalage dans le suivi et un plantage complet de l'application (obligation de rebooter)

Travail à réaliser : comprendre et supprimer ce bug de l'appli...

Autre idée : comment faire, si le performeur commet une grosse erreur (oubli d'une séquence, nombreuses fausses notes consécutives), mais se rattrape ensuite, pour recaler "à la main" le système de suivi qui se serait perdu ?

A.3 Réunion du 12 juin

Présents à la réunion : Serge Lemouton, Philippe Manoury, Diemo Schwarz, Gilles Mathieu.

Thème de la réunion : Discussion sur l'avancée du travail sur le suivi de partition, questions, remarques.

Résumé : Des dates sont plus ou moins fixées pour effectuer des tests avec un pianiste et une chanteuse. Ces tests seront probablement réalisés durant la première semaine de juillet.

Quelques idées générales et problèmes évoqués pour le suivi :

- Que se passe-t-il si la performance est jouée "hors tempo" ? Il faudrait définir une notion "d'intervalle minimum" entre deux notes déclenchant des cues, sans pour cela être trop rigide sur la durée des notes entre celles-ci. En effet, Philippe pense que donner une durée pour chaque note est trop contraignant pour le compositeur, qui peut souhaiter laisser le champ libre à son interprète. Diemo fait remarquer qu'une gestion du temps "souple" sur chaque note revient au même (durée modélisée par une courbe type gaussienne centrée sur la durée notée dans la partition).

- Problème des notes "muettes" : Concerne surtout le piano. si les touches sont enfoncées très doucement, un événement est-il déclenché ? Peut-on parler d'une vitesse limite ?

- Reconnaissance de contour : Philippe est très attaché à un développement dans ce sens. Diemo propose, sur certains passages, de rendre le suiveur plus tolérant : il doit reconnaître une note à plus ou moins un intervalle près, ce qui permet (notamment sur les passages rapides au chant) de détecter correctement un arpège, même pas très juste.

- Interface graphique : Développer une interface graphique apparaît comme l'une des attentes majeures dans le domaine du suivi. Cela suppose :

De fixer le mode de représentation

De fixer le protocole de données utilisées (les 2 sont très liés)

De trouver une personne qui s'en occupe...

Annexe B

Historique du travail réalisé

07-05

Meeting with Serge Lemouton and Philippe Manoury about the score following (see report for this meeting)

14-05

First documentation written

15-05

Encounter with Andrew Gerzso about the midi Score Following (suivimidi) (see report for this meeting)

22-05

New version of suivimidi.c, written by Nicola Orio Changes :

- Const **INT_TICK** is set to 5ms (instead of 10ms in previous version)
- the probability of the attack of a note is directly modeled

27-05

Changes in suivimidi.c :

In function "obscale", `ev->osbli[2] = prob_s;` becomes : `ev->osbli[2] = prob_r;`
(this wasn't a good idea, I changed it back after a few tests)

29-05

Changes in suivimidi.c :

- Enter Parameter for jMaxObject "suivimidi" is a list of 2 elements pitch, velocity instead of only the pitch as a single Integer, as previously
- Changes in functions "**suivimidi_newnote**" and "**suivimidi_update**" linked to the changes on parameter type.

30-05

Changes in `suivimidi.c` :

- A variable "**newnote_occured**" is added. It is set to 1 if a noteon has occurred, and to 0 if this note has been handled
 - Function **suivimidi_update** only makes update if **newnote_occured==1**
-

03-06

var **newnote_occured** is no more global, but member of struct **suivimidi_t**. This var has no more only binary value. It is incremented when a new note has occurred and decremented when a note has been handled.

a mode "nozigzag" has been written by Diemo Schwarz to prevent the follower from backward steps in the score.

05-06

Test in **suivimidi_refill** to prevent Segmentation faults by accessing an array with negative index (case of states REST, on which `poly=0` and `poly-1` is negative)

loop in **suivimidi_update** to highlight all notes in chords in the score instead of only one.
variables added in debug mode for statistic purpose (counters and quality indicators)

06-06

CVS commit for actual revision of `suivimidi.c`

07-06

CVS commit for all the tests-files used for `suivimidi`.

10-06

Flag for debug verbose added. Level of verbose can be set in the patch, by sending messages to `suivimidi` object.

11-06

Changes in `suivimidi`

- **suivimidi_newnote** : flag "isnew" is set to 1 only when new event is a noteon
 - **suivimidi_update** : when the follower is in a g-state, object `suivimidi` outputs now the pitch of the related note, and outputs it only one time
 - **suivimidi_obscalc** : "**prob_r**" is set to "**rest_ok**" when note velocity is 0 (noteoff) (in previous versions, was : when note pitch is 0)
-

12-06

Meeting with Serge Lemouton and Philippe Manoury about the score following (see report for this meeting)

Function **suivimidi_update** rewritten :

- better structure
 - algorithm optimized
 - comments cleaned
-

13-06

Creation of object suivieval, to make statistic on score following results

17-06

CVS commit suivieval.c

first version that compiles...

21-06

in suivieval.c :

setfilename bug fixed

output log files more pretty and "readable"

error-threshold settable in patch

24-06

in suivieval.c :

- realloc bug fixed : number of events to be recorded is now fixed from the beginning, and settable by user. If there are more events than space allowed, events are not recorded but there is no bug

- a 4th file is written, with informations of the match.log file, statistical datas (mean delta, std delta, number of errors, % recognition...) and informations on test conditions (modes, error-threshold)

24-06

in suivimidi.c :

- trills are modeled by a chord in a different midi channel

27-06

new jMax object created : suivimakecue

used to add cues to a reference note track

01-07

object suivieval outputs a list of statistic variables at 1st outlet

03-07

Object suivimakecue finished. This object record the label track for a given note track

04-07

files suivimakenet.c and suivimakenet.h created.

Modeling Function for score parsing are now in those files instead of suivimidi.

05-07

Help files for score following classes written and added to package

08-07

in suivimakenet.c :

- functions added to factorize calculations
 - a gstate (type **g_rest**) is added after each rest-state
 - **max_diff** is a parameter settable by user in suivimidi.
-

09-07

in suivimakenet.c :

G-rests are modeled before corresponding rests

10-07

in suivimakenet.c :

Model changed : ghost-states (g-states and g-rests) are now modeled before their corresponding n-state or rest Map of allowed transitions changed.

11-07 -> 17-07

in suivimakenet.c :

Model structure changed :

- A state is made at each change in the score (very useful in case of polyphonic events with overlaps)

- There are 6 types of changes :

- . "One_attack" : the only one event within **max_diff** is a note begin ;
 - . "One_release" : the only one event within **max_diff** is a note end ;
 - . "Poly_attack" : 2 or more notes begin within **max_diff** ;
 - . "Poly_release" : 2 or more notes end within **max_diff** ;
 - . "Legato" : 1 note ends and another begin within **max_diff** ;
 - . "Poly_legato" : 2 or more notes end and 1 or more begin within **max_diff** ;
 - In each state which is not a rest, we have the current note and all alive notes
-

23-07 -> 30-07

midi perf entry matching adapted to new model :

- in suivimakenet.c :

. Changes in refill

. Alive notes handling when we make a refill

- in suivimidi.c :

. changes in **suivimidi_obscalc**

31-07 -> 01-08

New object suivimapper : gets labels from suiviaudio or suivimidi and outputs labels when needed

02-08 -> 06-08

Alive-notes handling debugged in suivimakenet.c

In suivimidi.c :

Attacked notes are considered more important than alive notes in matching.

07-08

In suivimidi : Added functions to force the follower to go forward or backward while following.

08-08

midi_demo.jmax and **suivimidi.help.jmax** updated

09-08 -> 12-08

In suivimakenet.c and .h :

New way to cut the sequence of events into a sequence of realevents :

the whole sequence is cut when we do the first makenet. This avoids a lot of problems :

- size problems (we are sure to have enough realevents to model our states)
 - oldrests and **old_alivenotes** problems (we don't care anymore)
-

13-08

- Functions for forward and backward forcing added to suivaudio
 - In suivimidi : a label can now be associated to a rest
-

14-08

In suivimakenet.c :

Factorised state creation in function "**suivimakenet_addstate**", that sets state values (time, duration...) and calls **suivimakenet_dostate**.

In suivimidi.c :

Attack_weight is now a parameter (default : **ATTACK_WEIGHT** = 0)

19-08 -> 23-08

Little changes and debug works.

Documentation update, cleaned directories

Annexe C

Screenshots

C.1	Patch d'aide de l'objet suiviaudio	40
C.2	Patch d'aide de l'objet suivimidi	41
C.3	Patch d'aide de l'objet suivimakecue	42
C.4	Patch d'aide de l'objet suivimapper	43
C.5	Patch d'aide de l'objet suivieval	44
C.6	suvieval utilisé avec suivimidi	45
C.7	suvieval utilisé avec suiviaudio	46

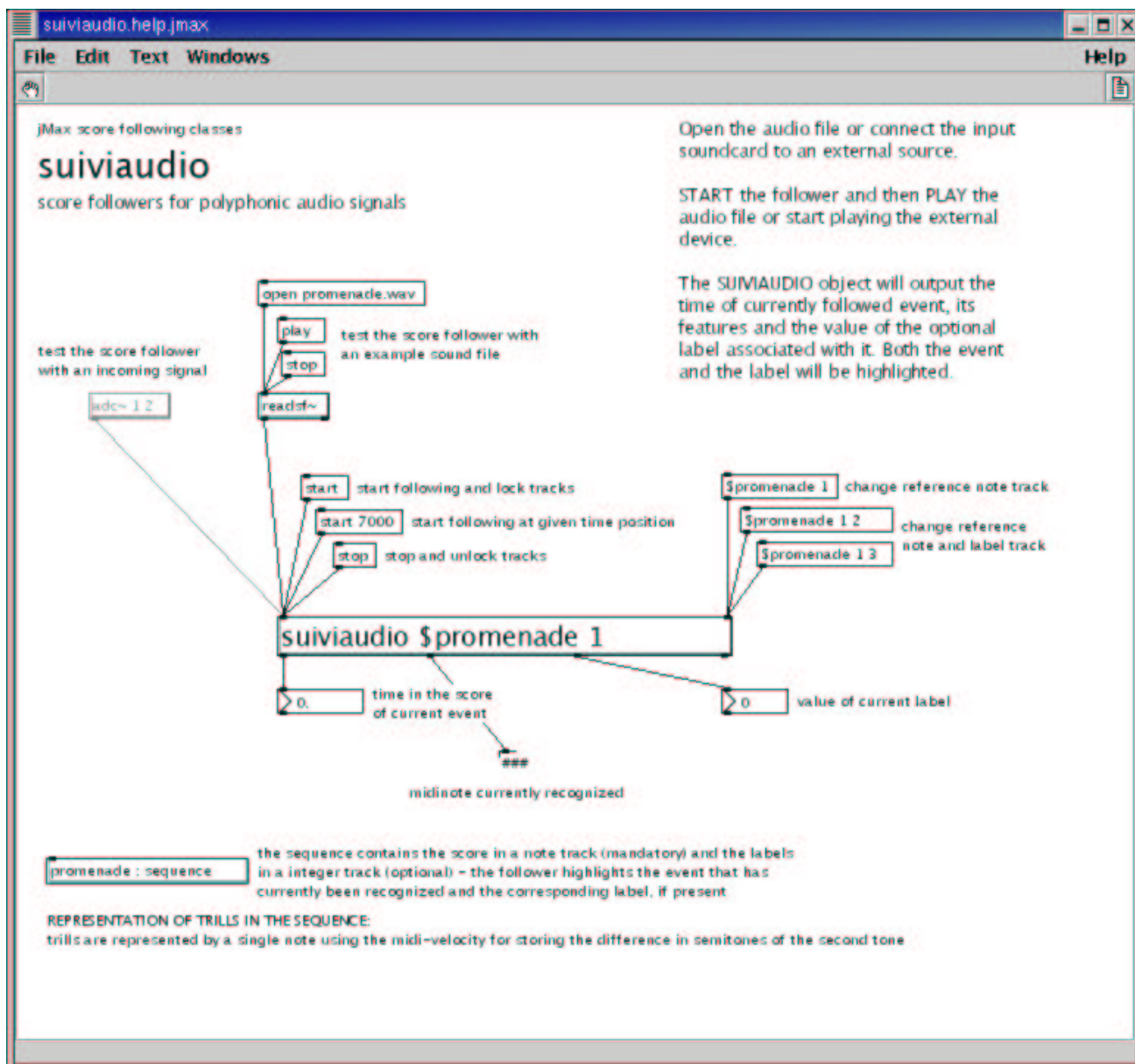


FIG. C.1 – Patch d'aide de l'objet suiviaudio

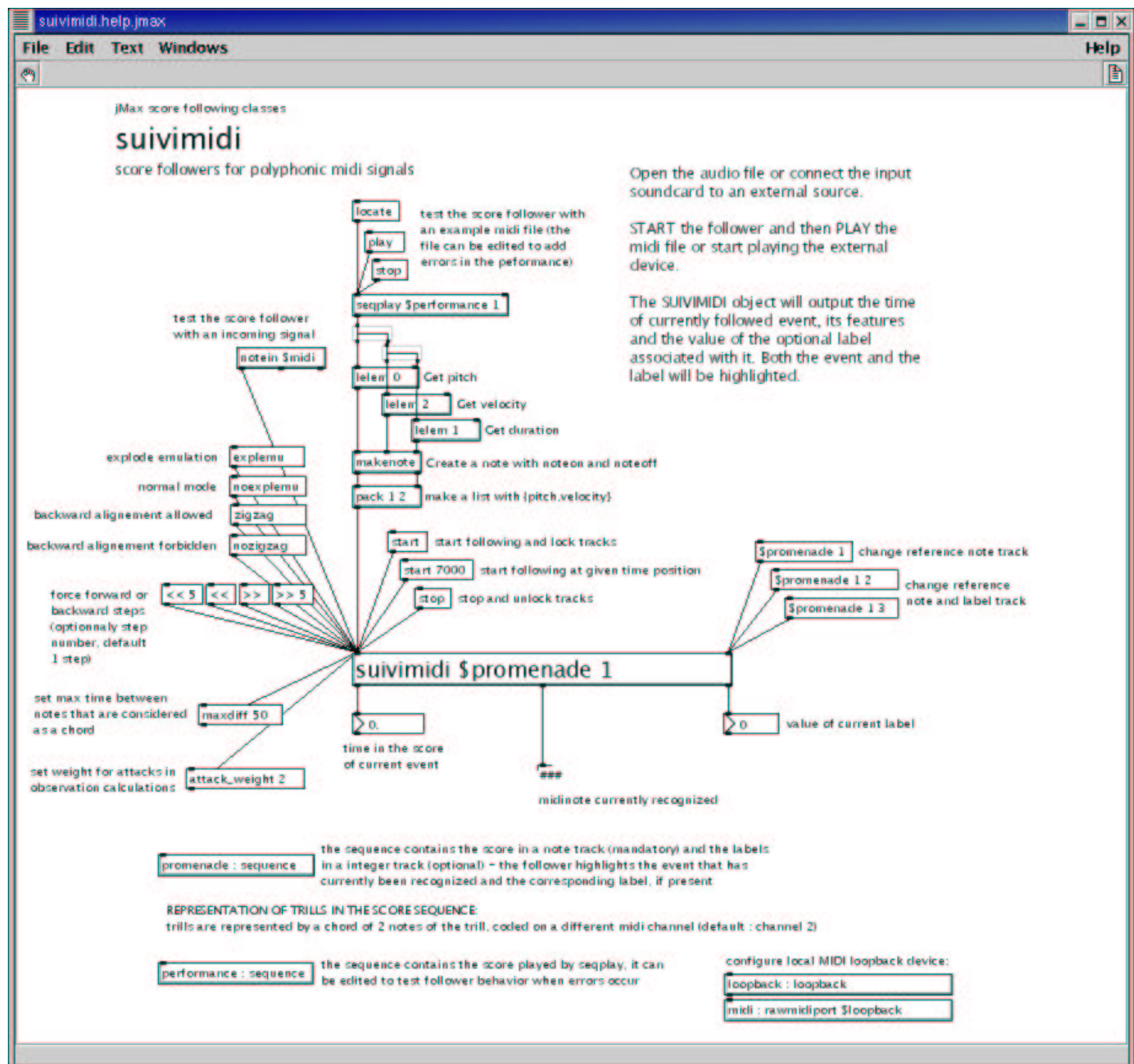


FIG. C.2 – Patch d'aide de l'objet suvimidi

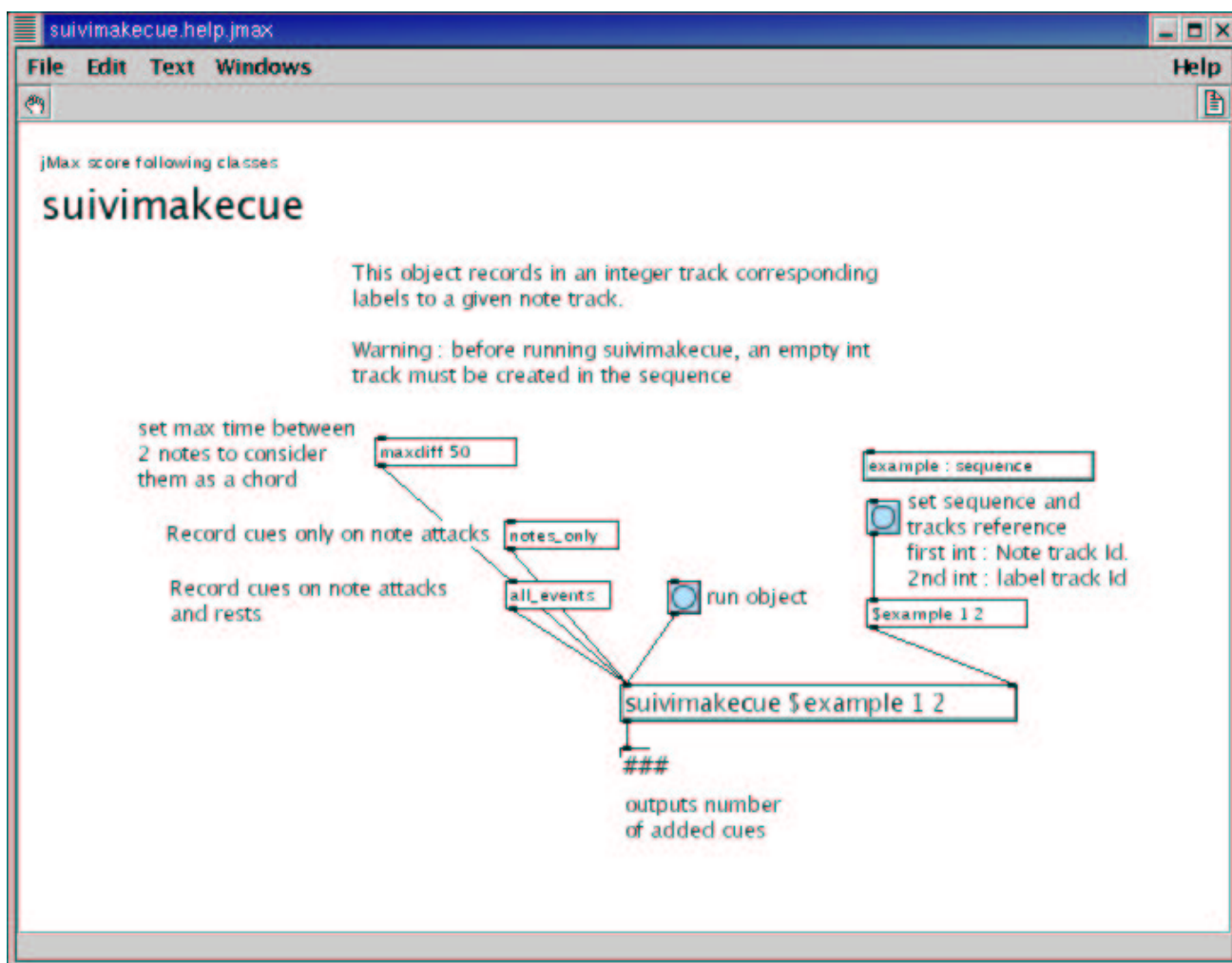


FIG. C.3 – Patch d'aide de l'objet suivimakecue

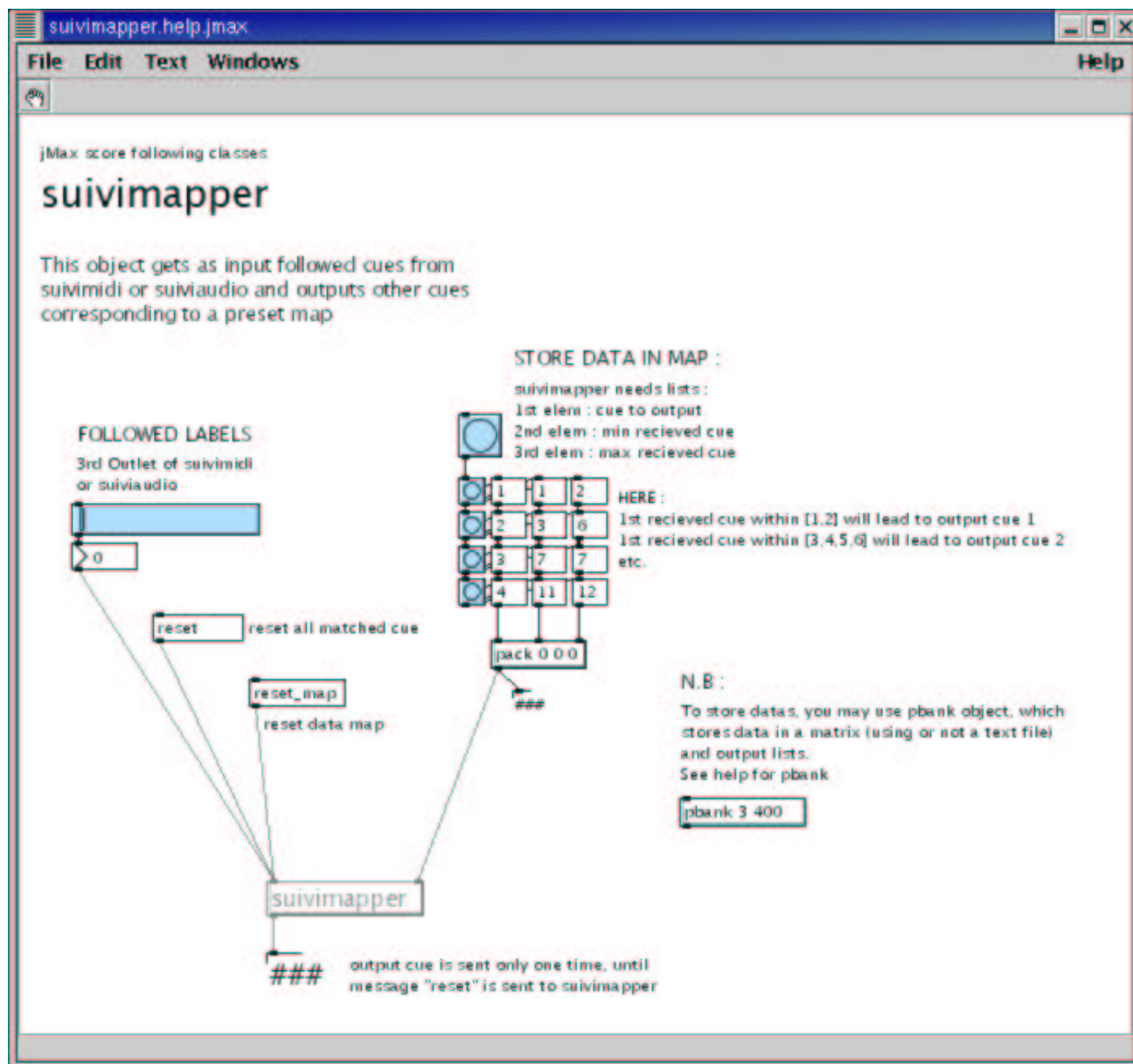


FIG. C.4 – Patch d'aide de l'objet suivimapper

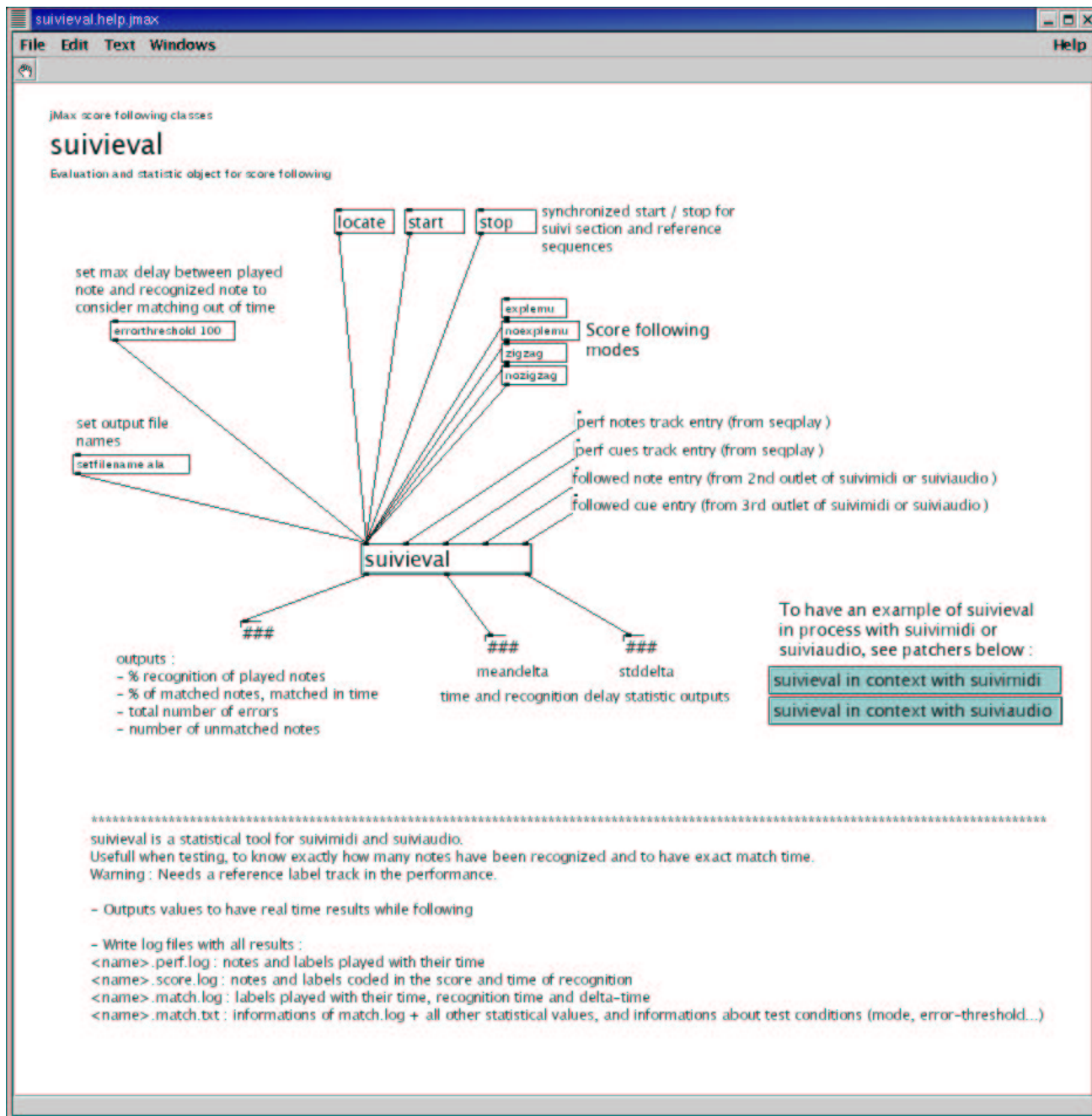


FIG. C.5 – Patch d'aide de l'objet suivieval

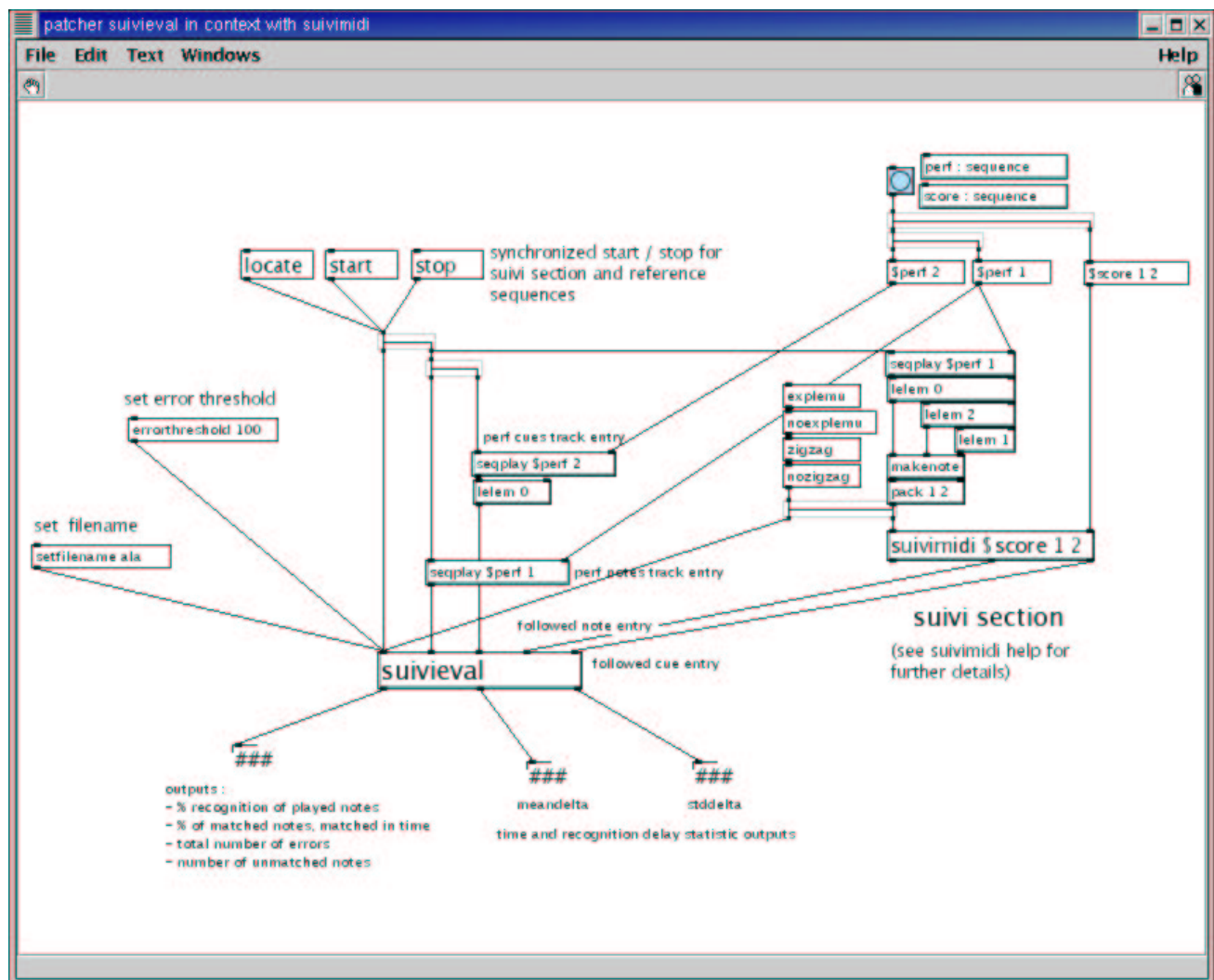
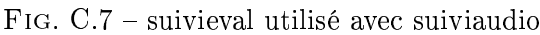


FIG. C.6 – suivieval utilisé avec suivimidi



Bibliographie

- [ALB99a] P. Cano A. Loscos and J. Bonada. Low-Delay Singing Voice Alignment to Text. In *Proceedings of the ICMC*, 1999.
- [ALB99b] P. Cano A. Loscos and J. Bonada. Score-Performance Matching using HMMs. In *Proceedings of the ICMC*, pages 441–444, 1999.
- [Bil98] J.A. Bilmes. *A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models*. ICSI, 1998.
- [C.R01] C.Raphael. In *Proceedings of the ICMC*, Havana, Cuba, 2001.
- [LCB⁺99] A. Loscos, P. Cano, J. Bonada, M. de Boer, and X. Serra. Voice Morphing System for Impersonating in Karaoke Applications. In *Proceedings of the ICMC*, 1999.
- [Mou01] V. Mouillet. *Prise en compte des informations temporelles dans les Modèles de Markov Cachés*. 2001.
- [OD01] N. Orio and F. Déchelle. In *Proceedings of the ICMC*, Havana, Cuba, 2001.
- [OS01] N. Orio and D. Schwarz. Alignment of Monophonic and Polypophonic Music to a Score. In *Proceedings of the ICMC*, Havana, Cuba, 2001.
- [Rab89] L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2) :257–285, 1989.
- [Rap99] C. Raphael. Automatic Segmentation of Acoustic Musical Signals Using Hidden Markov Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(4) :360–370, 1999.