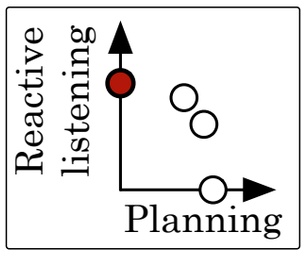
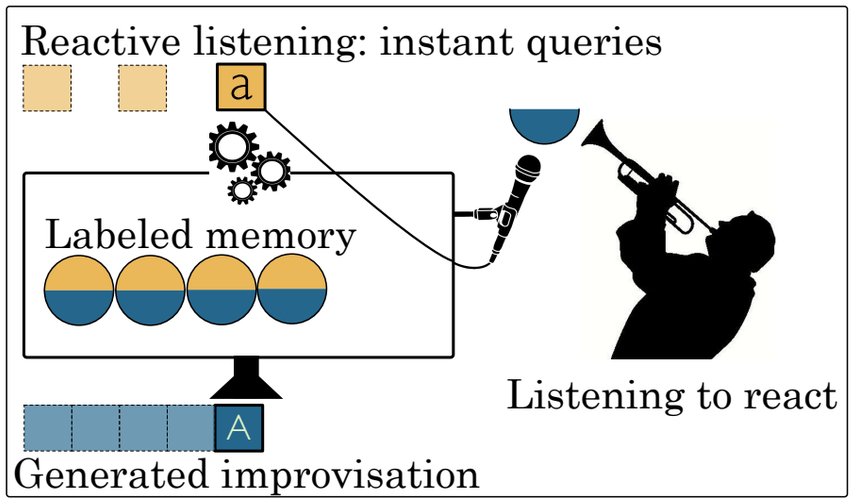
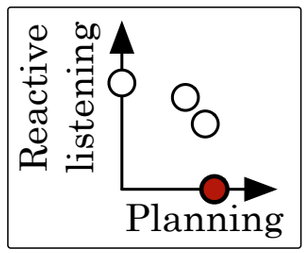
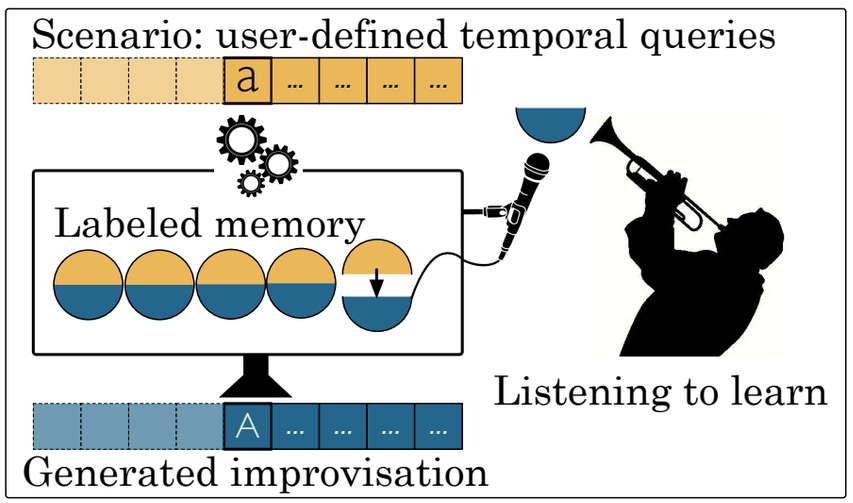




Label / features

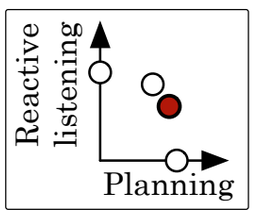
Music slice

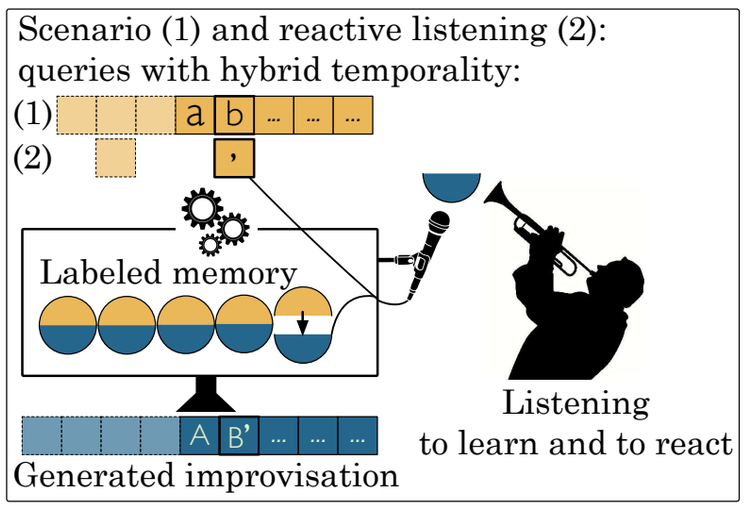




Label / features

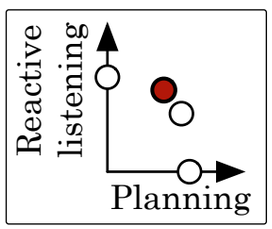
Music slice

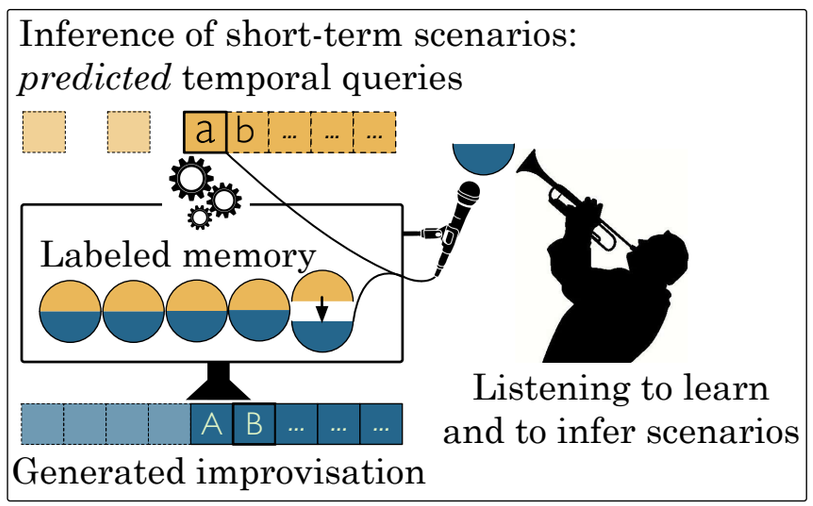


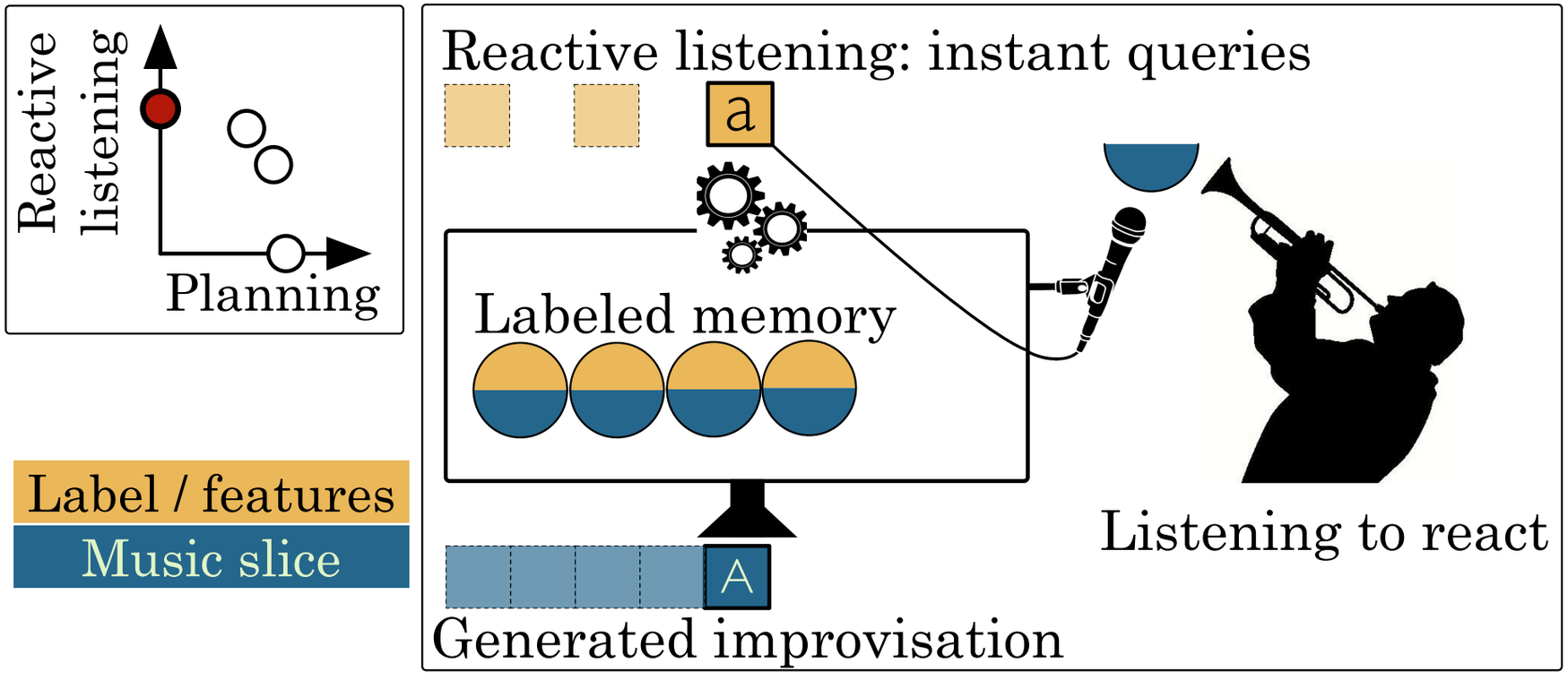


Label / features

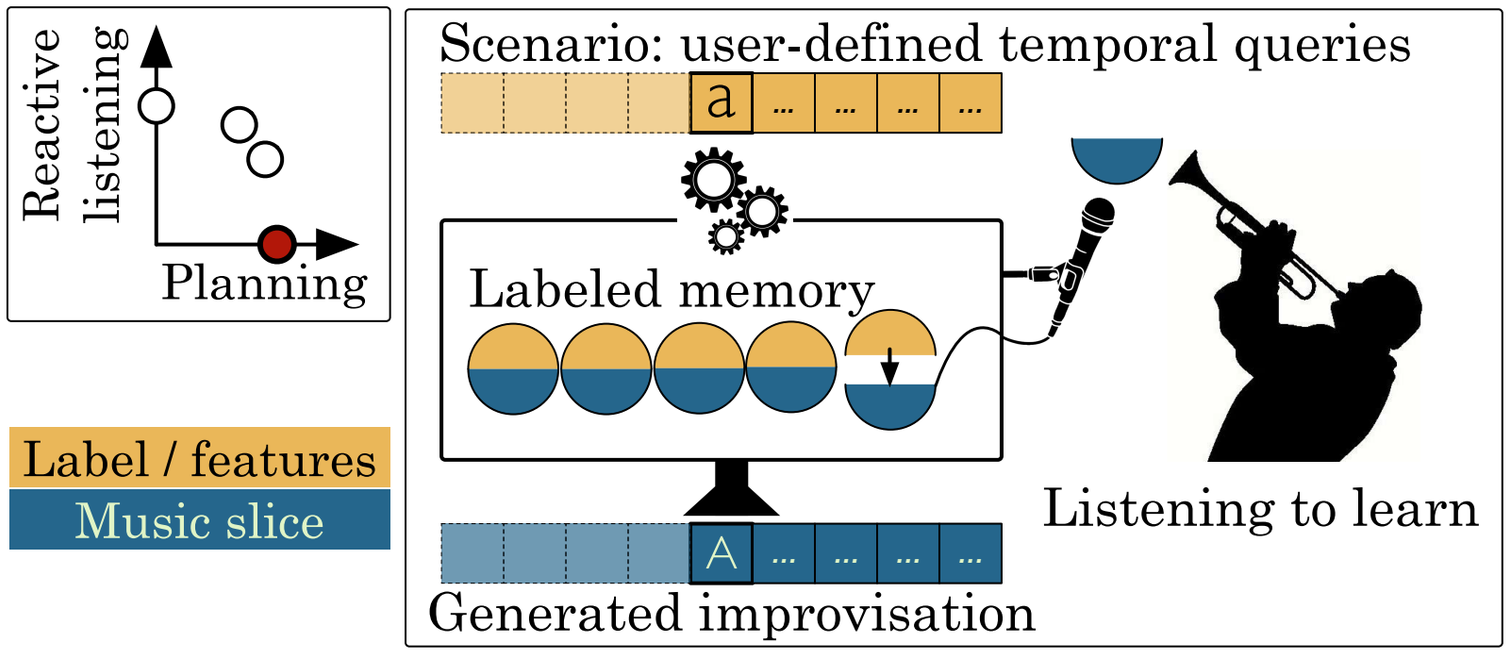
Music slice



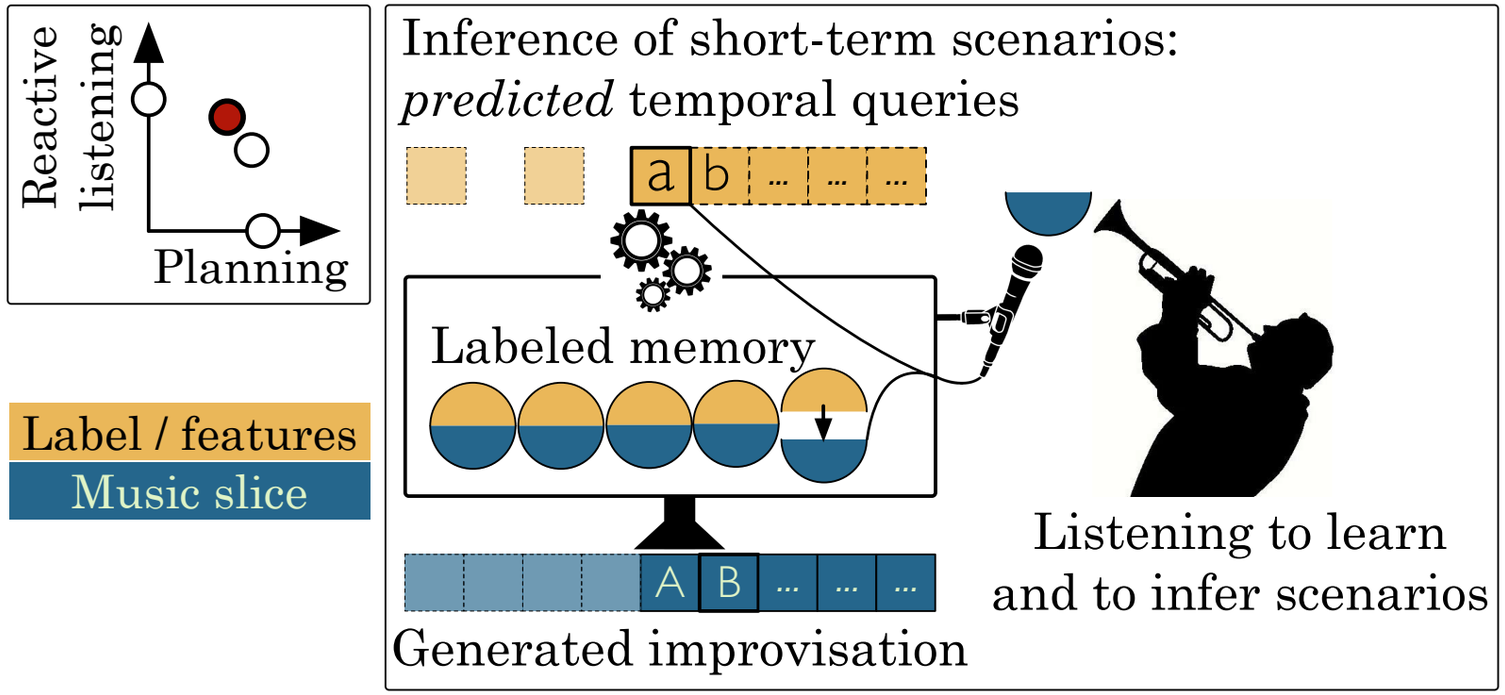



Reactive listening
Planning
Reactive listening: instant queries
a
Labeled memory
Listening to react
Label / features
Music slice
A
Generated improvisation

~ Somax



~ Prototype DYCI2 / ImproteK réactif +
requêtes court terme scénario réactif



—> Stage / thèse Tristan Carsault

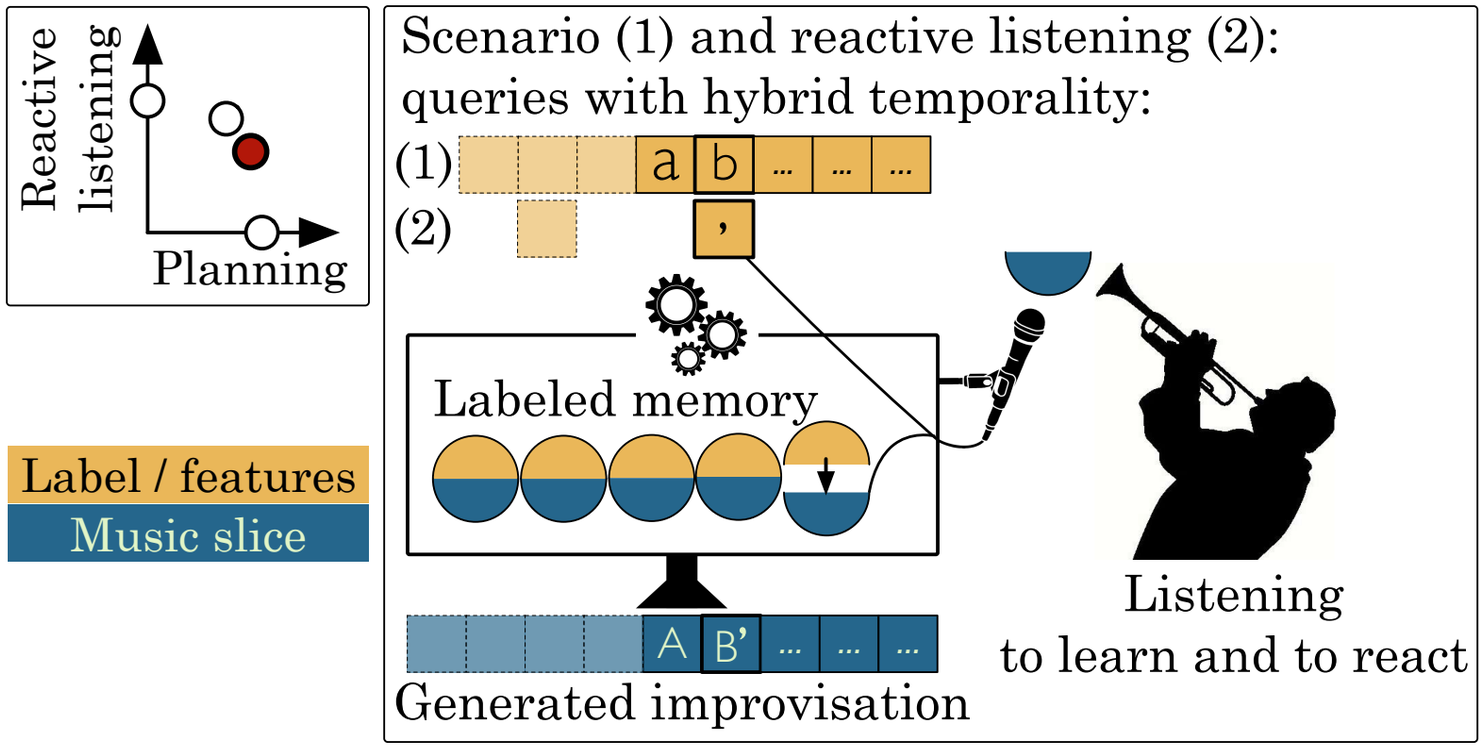

Reactive listening
Planning
Label / features
Music slice
Scenario (1) and reactive listening (2):
queries with hybrid temporality:
(1)
a
b
...
...
...
(2)
,
Labeled memory
A
B'
...
...
...
Generated improvisation
Listening
to learn and to react

Schéma architecture commune DYCI2 - ImproteK - Somax
Version « simplifiée »

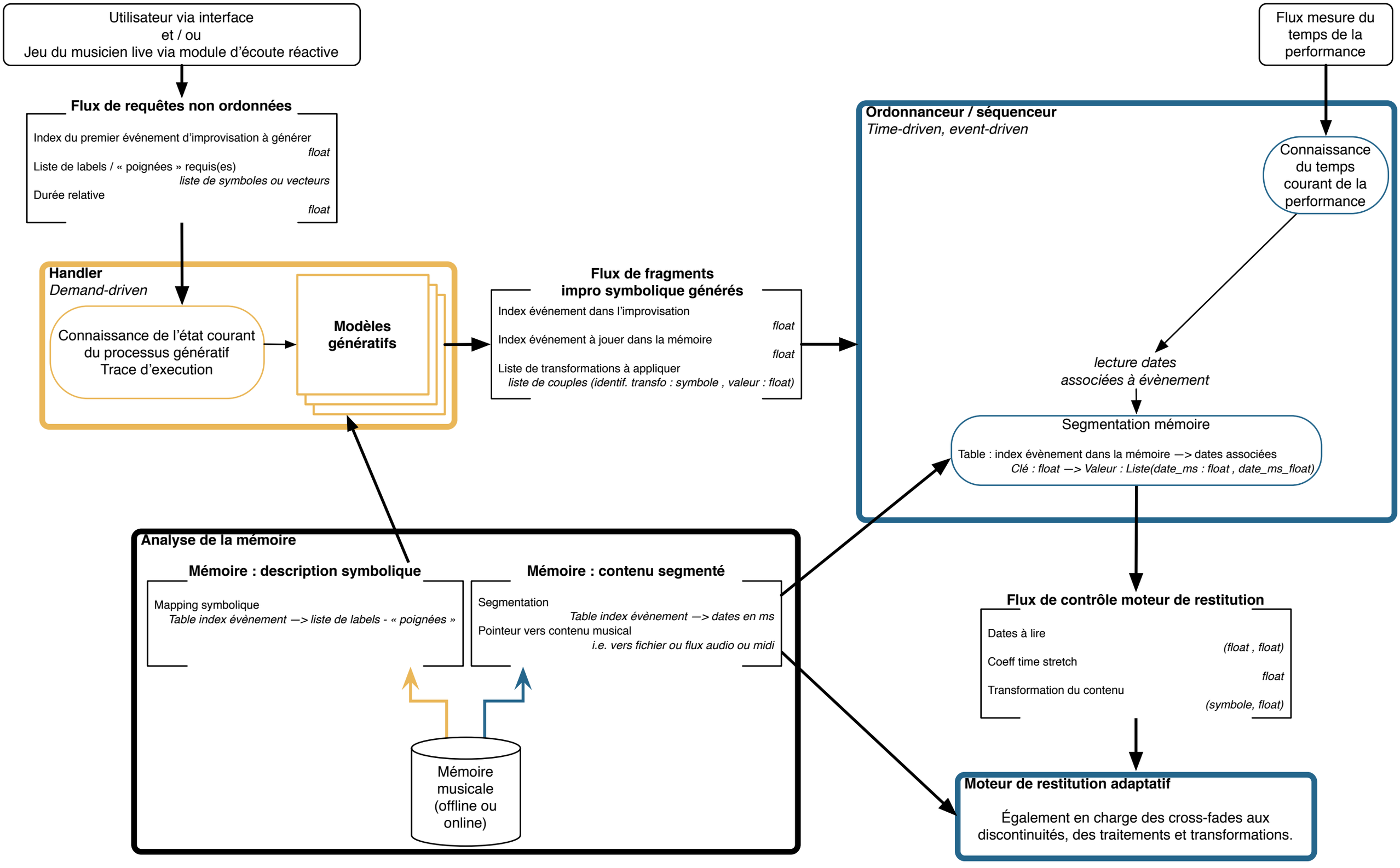


Schéma architecture commune DYCI2 - ImproteK - Somax
Version « simplifiée »

Utilisateur via interface
et / ou
Jeu du musicien live via module d'écoute réactive

Flux de requêtes non ordonnées

Index du premier événement d'improvisation à générer
float
Liste de labels / « poignées » requis(es)
liste de symboles ou vecteurs
Durée relative
float

Handler
Demand-driven

Connaissance de l'état courant du processus génératif
Trace d'execution

2

Modèles génératifs

1

Flux de fragments impro symbolique générés

Index événement dans l'improvisation
float
Index événement à jouer dans la mémoire
float
Liste de transformations à appliquer
liste de couples (identif. transfo : symbole , valeur : float)

Flux mesure du temps de la performance

Ordonnanceur / séquenceur
Time-driven, event-driven

Connaissance du temps courant de la performance

lecture dates associées à évènement

Segmentation mémoire

Table : index évènement dans la mémoire —> dates associées
Clé : float —> Valeur : Liste(date_ms : float , date_ms_float)

3

Analyse de la mémoire

Mémoire : description symbolique

Mapping symbolique
Table index évènement —> liste de labels - « poignées »

Mémoire : contenu segmenté

Segmentation
Table index évènement —> dates en ms
Pointeur vers contenu musical
i.e. vers fichier ou flux audio ou midi

Mémoire musicale (offline ou online)

Flux de contrôle moteur de restitution

Dates à lire
(float , float)
Coeff time stretch
float
Transformation du contenu
(symbole, float)

Moteur de restitution adaptatif

4

Également en charge des cross-fades aux discontinuités, des traitements et transformations.

Schéma architecture commune DYCI2 - ImproteK - Somax
Version « simplifiée »

Utilisateur via interface
et / ou
Jeu du musicien live via module d'écoute réactive

Flux de requêtes non ordonnées

Index du premier événement d'improvisation à générer
float
Liste de labels / « poignées » requis(es)
liste de symboles ou vecteurs
Durée relative
float

Handler
Demand-driven

Connaissance de l'état courant du processus génératif
Trace d'execution

2

Modèles génératifs

1

Flux de fragments impro symbolique générés

Index événement dans l'improvisation
float
Index événement à jouer dans la mémoire
float
Liste de transformations à appliquer
liste de couples (identif. transfo : symbole , valeur : float)

Flux mesure du temps de la performance

Ordonnanceur / séquenceur
Time-driven, event-driven

Connaissance du temps courant de la performance

lecture dates associées à évènement

Segmentation mémoire

Table : index évènement dans la mémoire —> dates associées
Clé : float —> Valeur : Liste(date_ms : float , date_ms_float)

3

Analyse de la mémoire

Mémoire : description symbolique

Mapping symbolique
Table index évènement —> liste de labels - « poignées »

Mémoire : contenu segmenté

Segmentation
Table index évènement —> dates en ms
Pointeur vers contenu musical
i.e. vers fichier ou flux audio ou midi

Mémoire musicale (offline ou online)

Flux de contrôle moteur de restitution

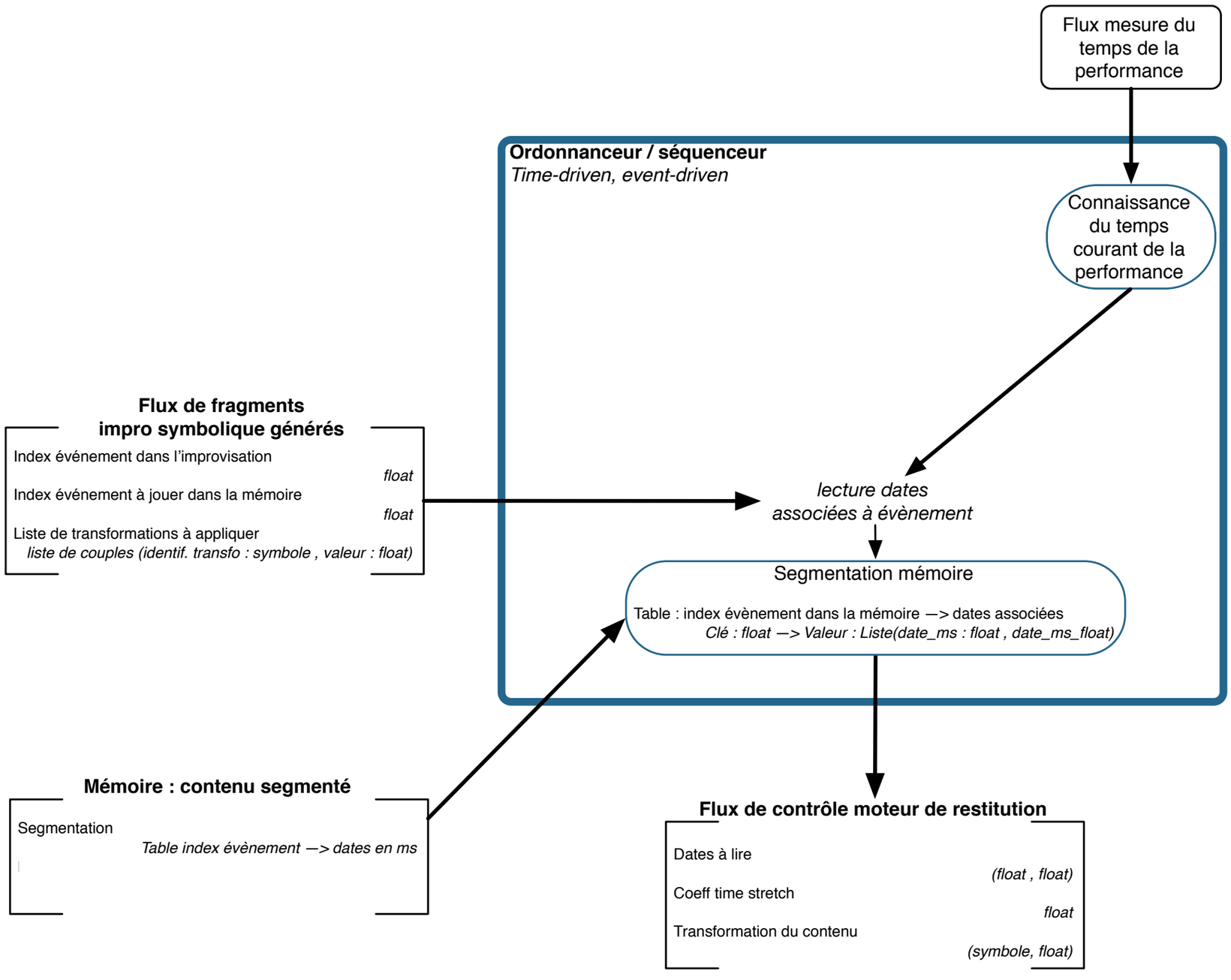
Dates à lire
(float , float)
Coeff time stretch
float
Transformation du contenu
(symbole, float)

Moteur de restitution adaptatif

4

Également en charge des cross-fades aux discontinuités, des traitements et transformations.

Ordonnanceur / séquenceur



- **Description** : ordonnanceur/séquenceur faisant le lien entre l'environnement et le temps de la performance, et les processus génératifs.
  Fait la transition entre le temps symbolique et le temps réel en permettant l'adaptation temporelle.

- **Entrées :**

  - Flux de fragments impro symbolique (ou plus généralement mapping dynamique évènements de la performance —> évènement de la mémoire)

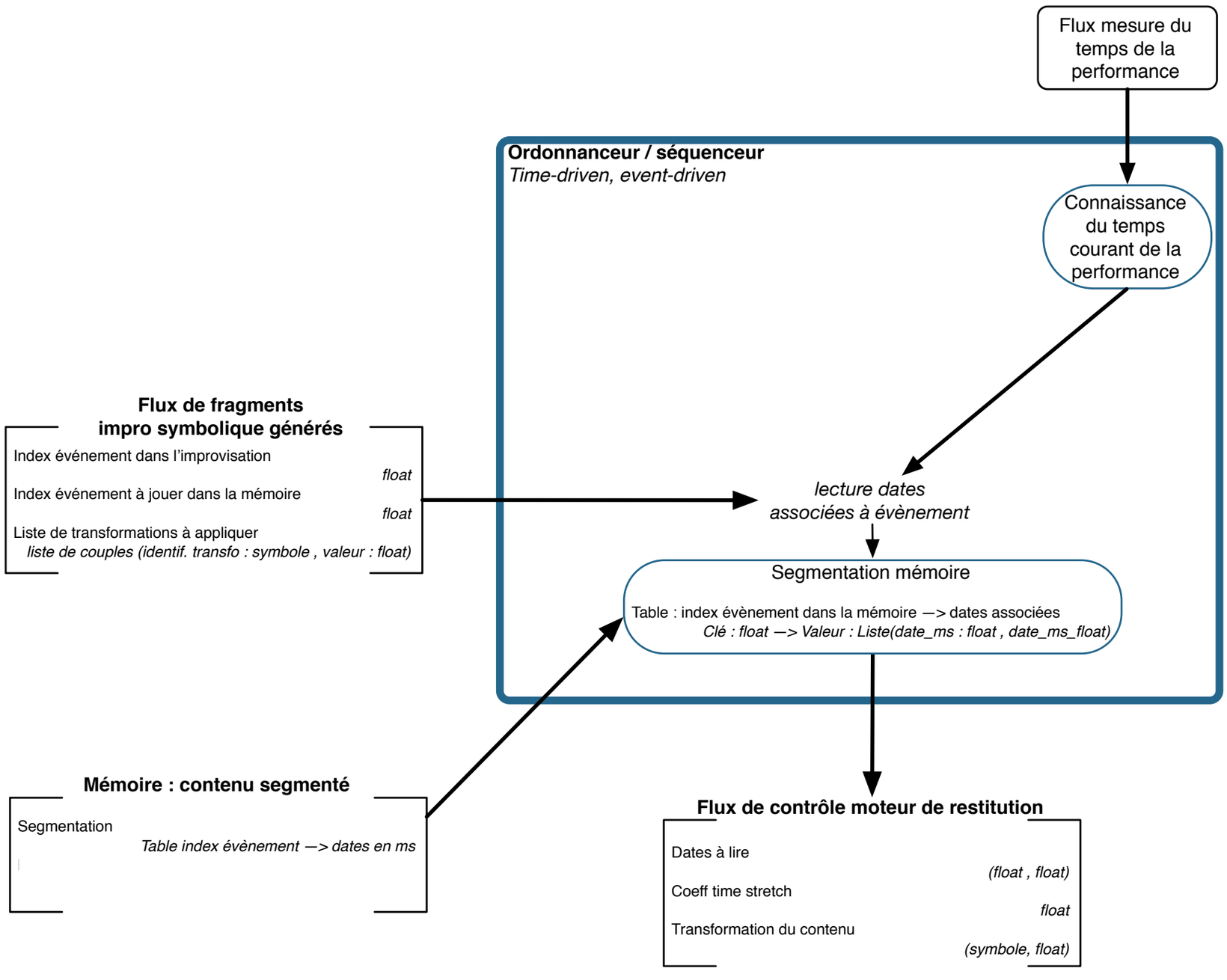
  - Données - Mémoire

  - Flux de mesure du temps de la performance : flux servant de référence temporelle mesurée en beats, en évènements,… cf. plus loin.

- **Sorties :**

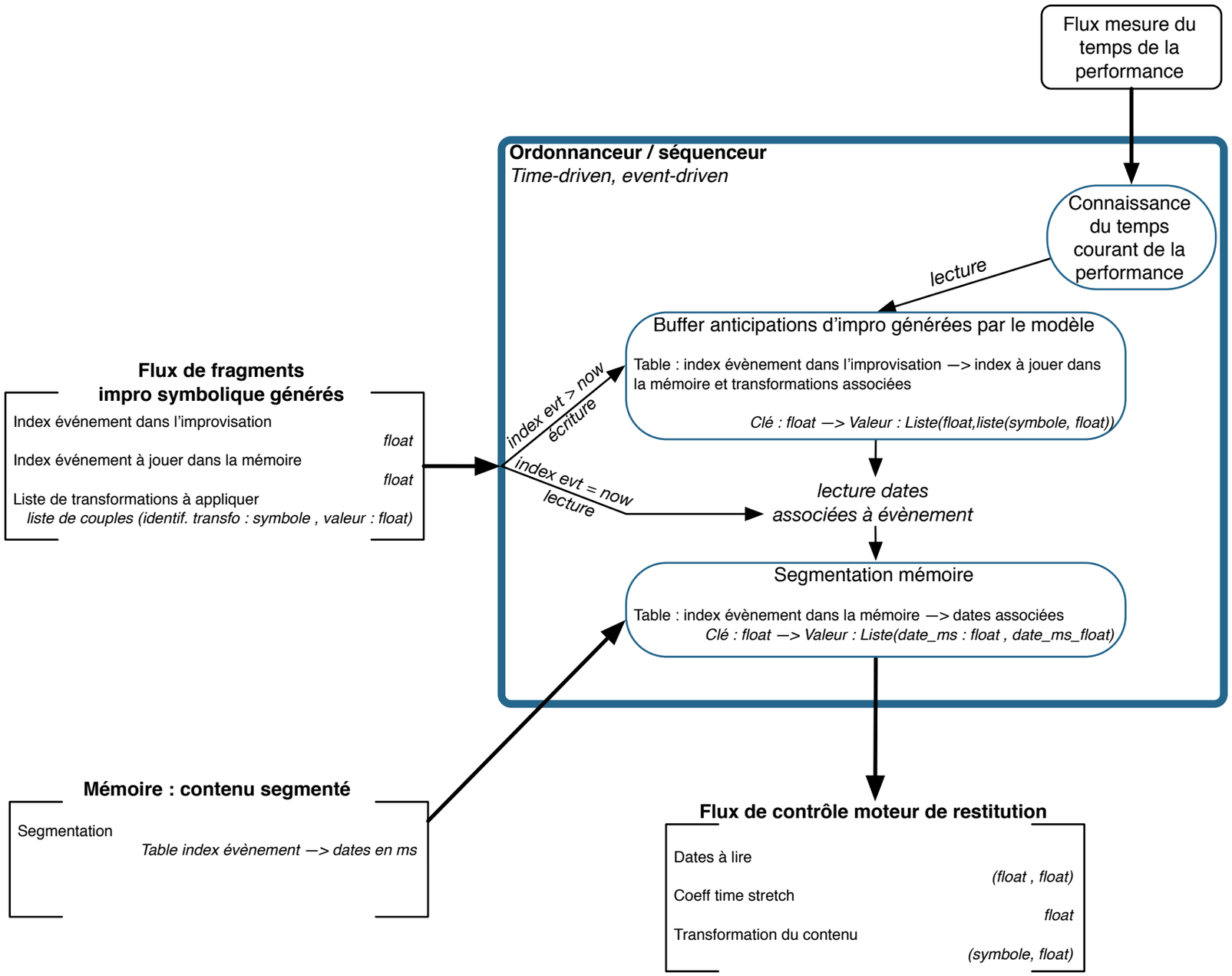
  - Flux de contrôle du moteur de restitution

Ordonnanceur / séquenceur



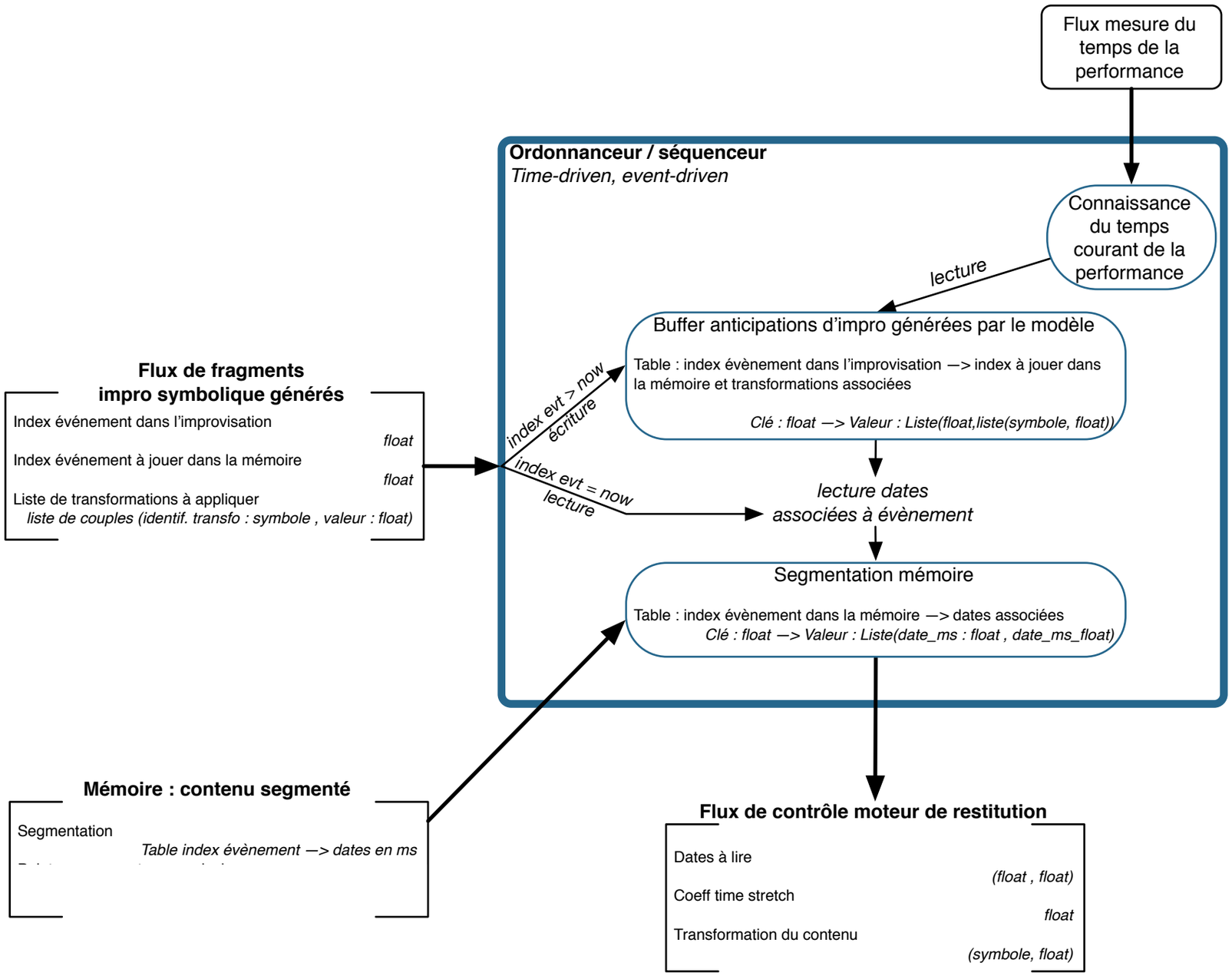
- **Remarque** : l'entrée est ici appelée « flux de fragments impro symbolique générés » à cause du contexte général dans lequel ce module s'inscrit.
  Plus généralement il s'agit simplement d'un « mapping dynamique évènements de la performance —> évènement de la mémoire ».
  Ce mapping pourrait en effet venir d'autre part qu'un processus génératif, par exemple d'une interface manuelle (cf. le « side-project » d'outil basique mais pratique dont j'ai déjà parlé plusieurs fois).

Ordonnanceur / séquenceur

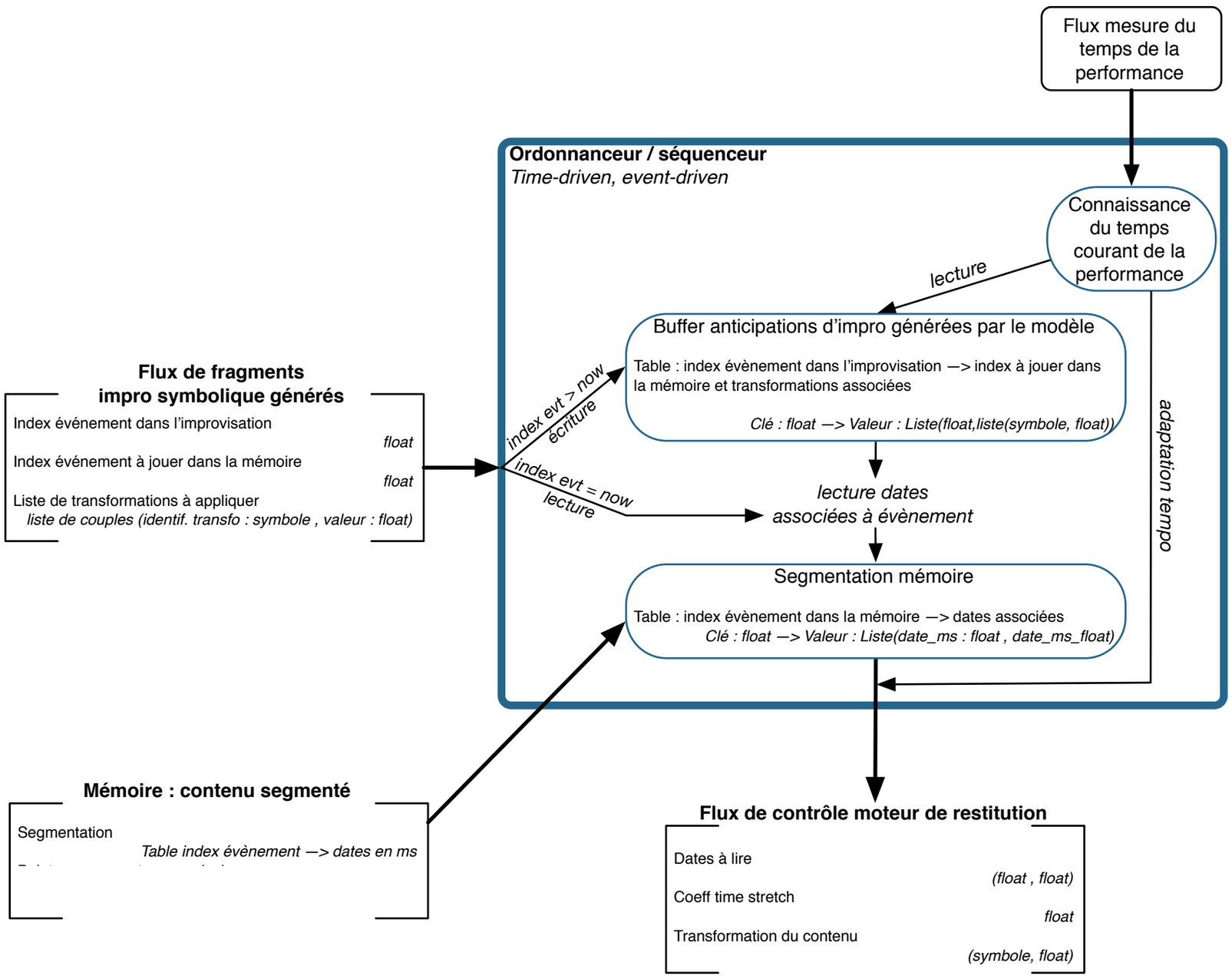


- **Anticipation / réaction** :
  - Dans un premier temps pour une utilisation soit 100 % « à la ImproteK » soit 100% « à la Somax » :
    - *Si évènement reçu concerne une date > à la date courante de la performance :*
      Il s'agit d'une anticipation (qui sera potentiellement réécrite), mise dans un buffer, lue en temps et en heure.
    - *Si évènement reçu concerne une date = à la date courante de la performance :*
      lecture immédiate
  - Dans un second temps pour « merger » les paradigmes
    - travail à effectuer plutôt au niveau du handler et des modèles
    - une première étape pourrait par contre se situer à ce niveau : non plus stocker *une* anticipation choisie dans le buffer, mais toutes, et de choisir la plus pertinente selon les infos provenant de l'écoute réactive (« guidage dur »). Voire choisir + transformer (premier pas vers processus « génératif » et non plus purement concaténatif ?)

Ordonnanceur / séquenceur

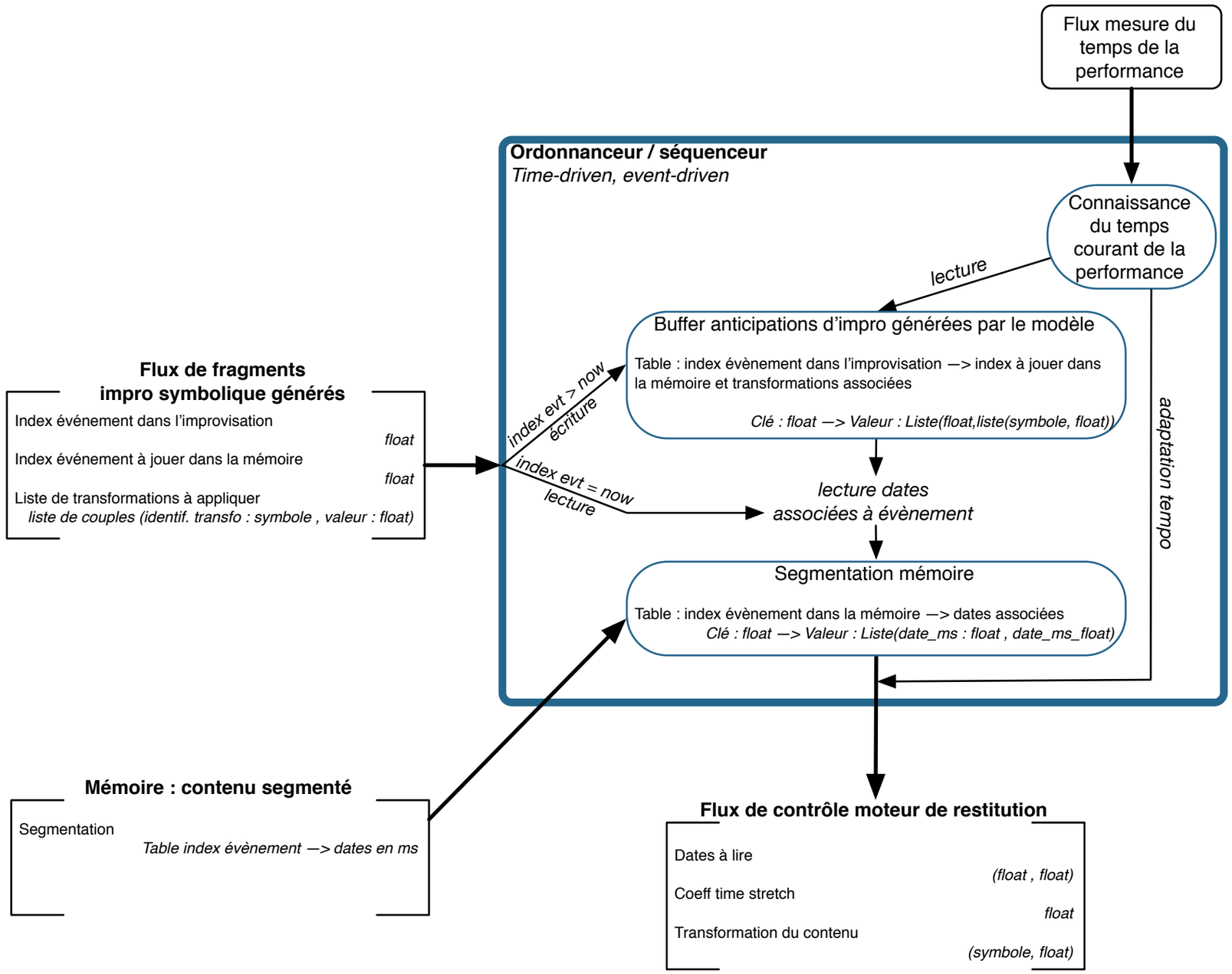


Ordonnanceur / séquenceur

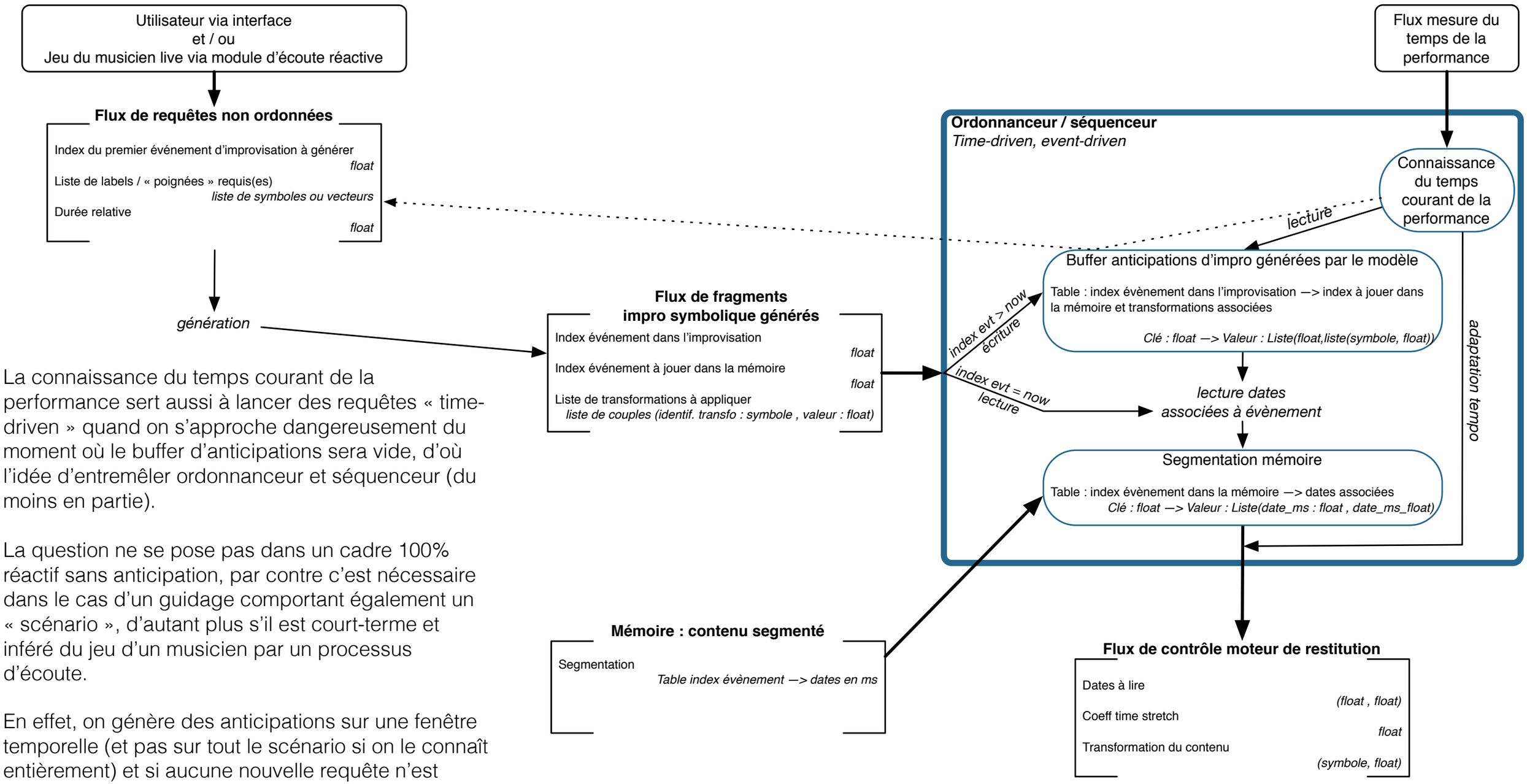


La connaissance du temps courant de la performance ne sert bien sûr pas uniquement à lancer la restitution (ou des requêtes, cf. plus loin) en temps et en heure, mais aussi à l'adaptation de la vitesse de la restitution par rapport au flux de mesure du temps de la performance.

Ordonnanceur / séquenceur



Ordonnanceur / séquenceur



La connaissance du temps courant de la performance sert aussi à lancer des requêtes « time-driven » quand on s'approche dangereusement du moment où le buffer d'anticipations sera vide, d'où l'idée d'entremêler ordonnanceur et séquenceur (du moins en partie).

La question ne se pose pas dans un cadre 100% réactif sans anticipation, par contre c'est nécessaire dans le cas d'un guidage comportant également un « scénario », d'autant plus s'il est court-terme et inféré du jeu d'un musicien par un processus d'écoute.

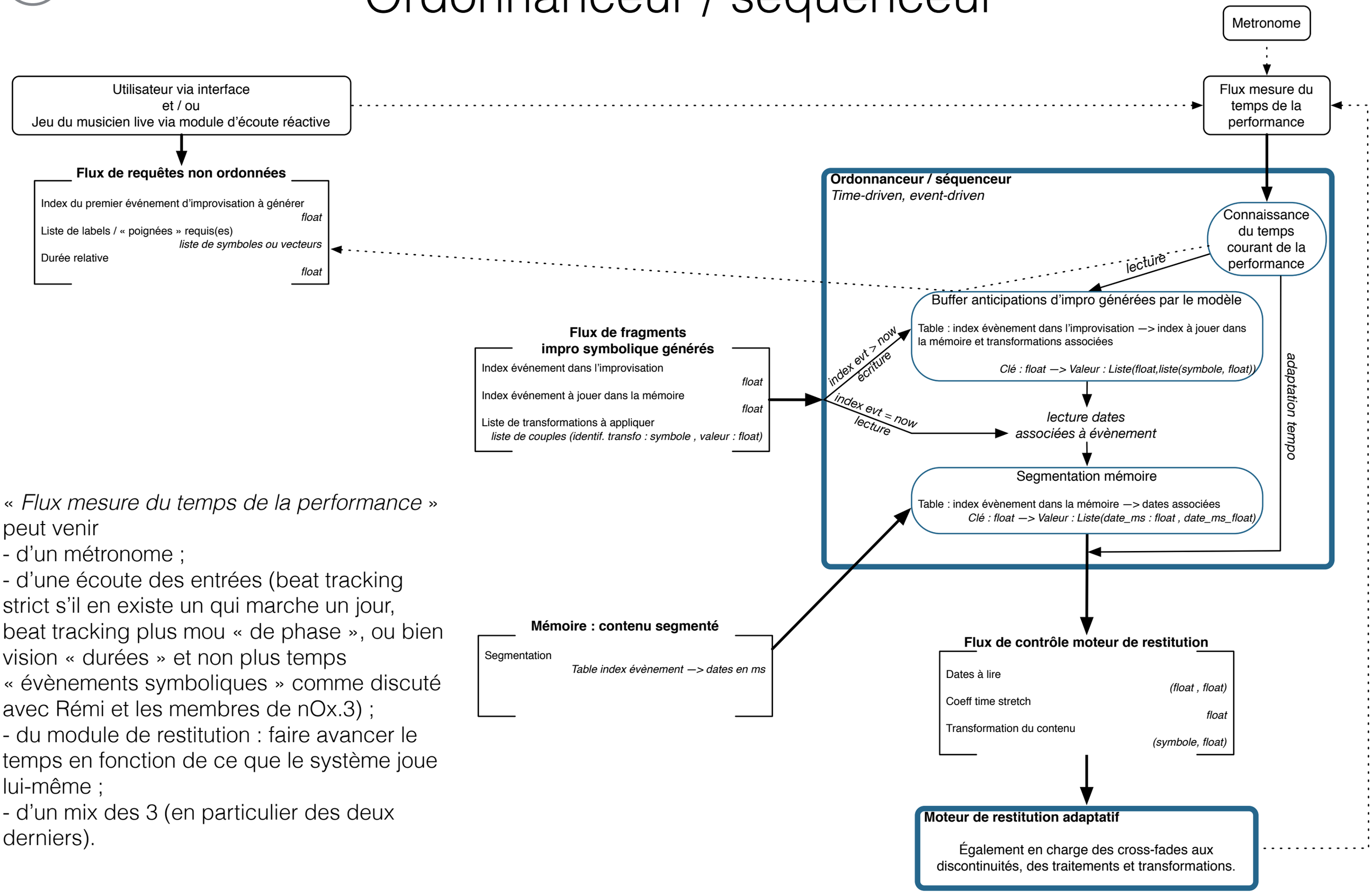
En effet, on génère des anticipations sur une fenêtre temporelle (et pas sur tout le scénario si on le connaît entièrement) et si aucune nouvelle requête n'est arrivée pour demander une nouvelle génération, il faut bien en relancer une avant qu'il n'y ait plus rien à jouer.

Peut également être intéressant dans un processus 100% « à la Somax » si on voulait tester d'autres modes de jeu que « recopier la mémoire telle quelle » entre deux appels au modèle, *i.e.* deux sauts.

3

Ordonnanceur / séquenceur

« *Flux mesure du temps de la performance* » peut venir
- d'un métronome ;
- d'une écoute des entrées (beat tracking strict s'il en existe un qui marche un jour, beat tracking plus mou « de phase », ou bien vision « durées » et non plus temps « évènements symboliques » comme discuté avec Rémi et les membres de nOx.3) ;
- du module de restitution : faire avancer le temps en fonction de ce que le système joue lui-même ;
- d'un mix des 3 (en particulier des deux derniers).



3

Ordonnanceur / séquenceur

Si besoin, le temps de la performance peut également être communiqué (seulement quand cela est nécessaire) à l'agent gérant la génération.

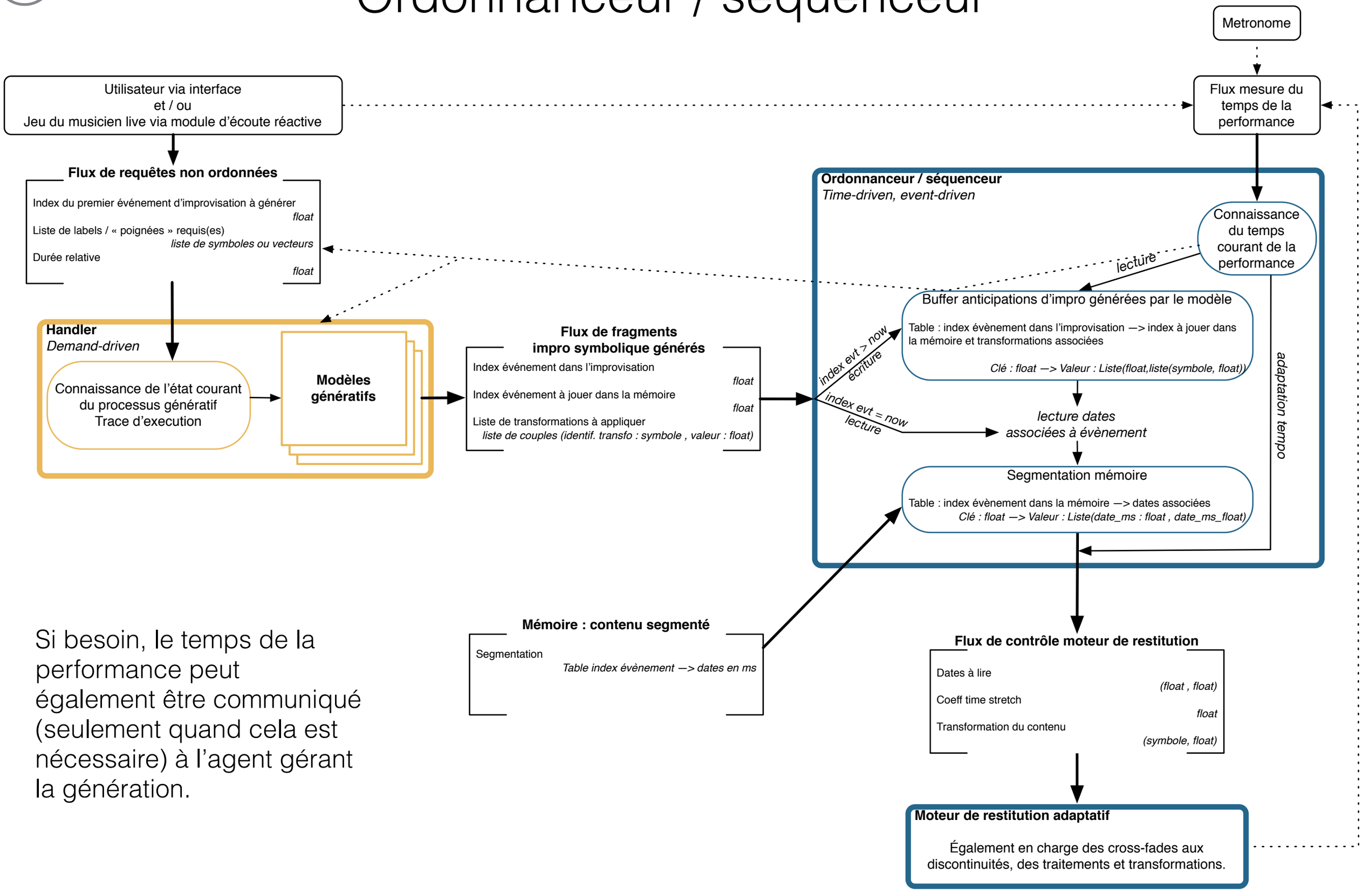


Schéma architecture commune DYCI2 - ImproteK - Somax
Version « simplifiée »

Utilisateur via interface
et / ou
Jeu du musicien live via module d'écoute réactive

Flux de requêtes non ordonnées

Index du premier événement d'improvisation à générer
float
Liste de labels / « poignées » requis(es)
liste de symboles ou vecteurs
Durée relative
float

Handler
Demand-driven

Connaissance de l'état courant du processus génératif
Trace d'execution

2

Modèles génératifs

1

Flux de fragments impro symbolique générés

Index événement dans l'improvisation
float
Index événement à jouer dans la mémoire
float
Liste de transformations à appliquer
liste de couples (identif. transfo : symbole , valeur : float)

Flux mesure du temps de la performance

Ordonnanceur / séquenceur
Time-driven, event-driven

Connaissance du temps courant de la performance

lecture dates associées à évènement

Segmentation mémoire

Table : index évènement dans la mémoire —> dates associées
Clé : float —> Valeur : Liste(date_ms : float , date_ms_float)

3

Analyse de la mémoire

Mémoire : description symbolique

Mapping symbolique
Table index évènement —> liste de labels - « poignées »

Mémoire : contenu segmenté

Segmentation
Table index évènement —> dates en ms
Pointeur vers contenu musical
i.e. vers fichier ou flux audio ou midi

Mémoire musicale (offline ou online)

Flux de contrôle moteur de restitution

Dates à lire
(float , float)
Coeff time stretch
float
Transformation du contenu
(symbole, float)

Moteur de restitution adaptatif

4

Également en charge des cross-fades aux discontinuités, des traitements et transformations.

4

Moteur de restitution adaptatif

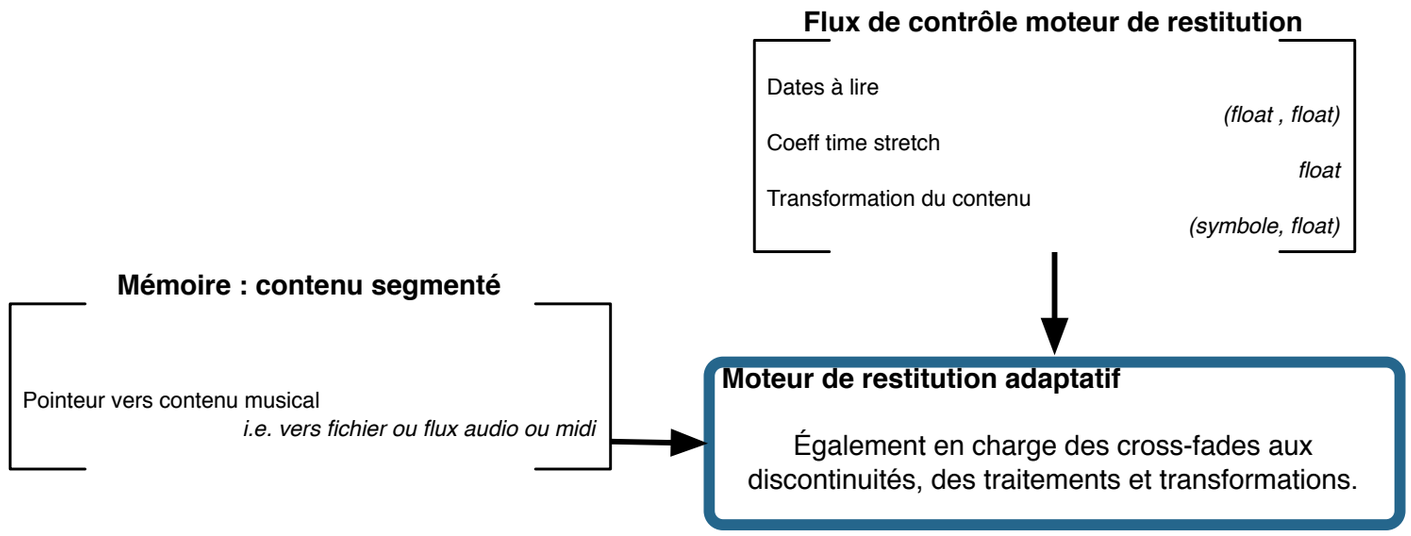
- **Description** :
  Le moteur de rendu n'a pas besoin de savoir si ce qu'on lui demande de jouer sur l'instant est le résultat d'un processus réactif, d'une génération offline… doit juste connaître des dates à lire et les transformations (time stretch ou autres) à appliquer au matériau.

- **Entrées :**
  - Données - mémoire : pointeur vers fichier(s) ou flux mémoire ;
  - Flux de contrôle : position à lire, vitesse de time stretch, transformation à appliquer au matériau (par exemple transposition, ajouter du gain,…)

- **Sortie :**
  - Restitution

- Sur le même modèle : un « renderer » audio, un midi, voire ensuite vidéo… ou contrôler autres paramètres de la restitution (effets, synthèse…). Si le reste est bien fait, peut être le seul module à dépendre du type des données qu'on manipule.

- En pratique :
  - contient autant de « players » que de voix dans l'improvisation,
  - chargé d'appliquer les transformations applicables au matériau (transposition, gain, etc.).



- **Nouvelle version :**
  - **Anticipation Xfade —> qualité audio,**
  - **Beat, évènement…**

ImproteK-InterfaceAudio (presentation)

ImproteK IT_1

p click

p printAnte

INPUTS

OnlineParam01

or

offline_only

p record

OUTPUTS

CURRENT IMPRO

Length scenario 0

Current cycle 0

Pos. in scenario 0

Pos. playing 0

Tempo 120

Original tempo * 0.

QUICK START
1) If necessary, set the input parameters in "OnlineParam01"
2) Audio on
3) Edit / Click a message in one of the scenario presets patches (grid_data = scenario, see the list in "p LOAD")
4) Edit the parameters of the 3 voices (rigth)
5) Click on GENERATE
6) Click on BEGIN

SCENARIO

p LOAD

p exp_ChangeScenario

p presets-Burger

p presets-Gentilucci

p presets-Fox

p presets-Delbecq

p HELP-PRESETS

GENERATION

GENERATE
with current parameters

Automatically launch a new generation at each new cycle

p buffers_and_controls

p memory_zone_online

TEMPO AND BEAT

Pulsed

Tempo 120.

p external_beat_midi

start_metro

p gridconnect

BEGIN

Beat

96 Voice1_cont

Online / Offline

Memory zone (cycles) 1 to 1

Taboos (events) 0 to 0

Use from X to last cycle

Use last 1 cycles

p LoadOfflineMemory

Offline buffer: None

New generation at next query

Transpo Δ 2

RT

No rep.

Player name: Voice1

On buffer: online_Buffer

Mono / Stereo

Pos. in buffer

Transf 0. 0.

Manual transpo 0.

Play 60 Duration crossfade (ms)

2 -1 1.017569 1.2

Voice1_offline

p peak_limiter

p compressor

96 Voice2_cont

Online / Offline

Memory zone (cycles) 1 to 1

Taboos (events) 0 to 0

Use from X to last cycle

Use last 1 cycles

p LoadOfflineMemory

Offline buffer: None

New generation at next query

Transpo Δ 2

RT

No rep.

Player name: Voice2

On buffer: online_Buffer

Mono / Stereo

Pos. in buffer

Transf 0. 0.

Manual transpo 0.

Play 60 Duration crossfade (ms)

2 -1 1.017569 1.2

Voice2_offline

p peak_limiter

p compressor

96 Voice3_cont

Online / Offline

Memory zone (cycles) 1 to 1

Taboos (events) 0 to 0

Use from X to last cycle

Use last 1 cycles

p LoadOfflineMemory

Offline buffer: None

New generation at next query

Transpo Δ 2

RT

No rep.

Player name: Voice3

On buffer: online_Buffer

Mono / Stereo

Pos. in buffer

Transf 0. 0.

Manual transpo 0.

Play 60 Duration crossfade (ms)

2 -1 1.017569 1.2

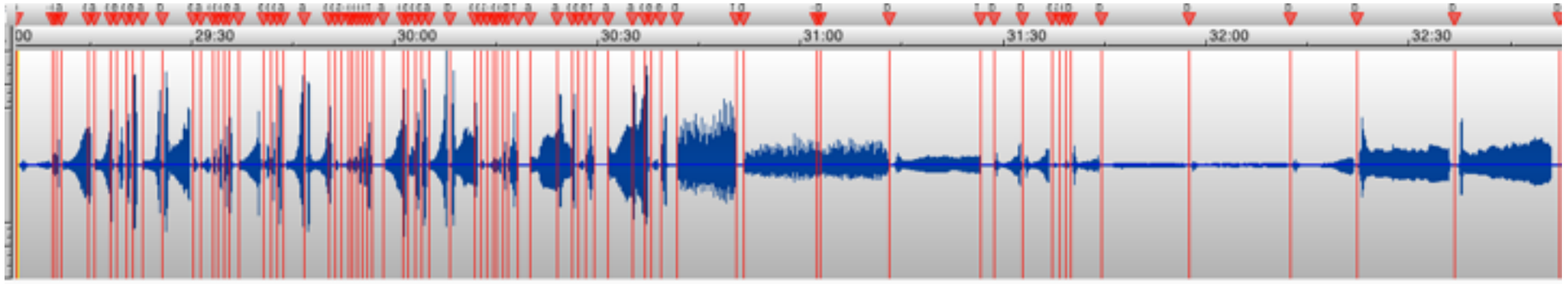
Voice3_offline

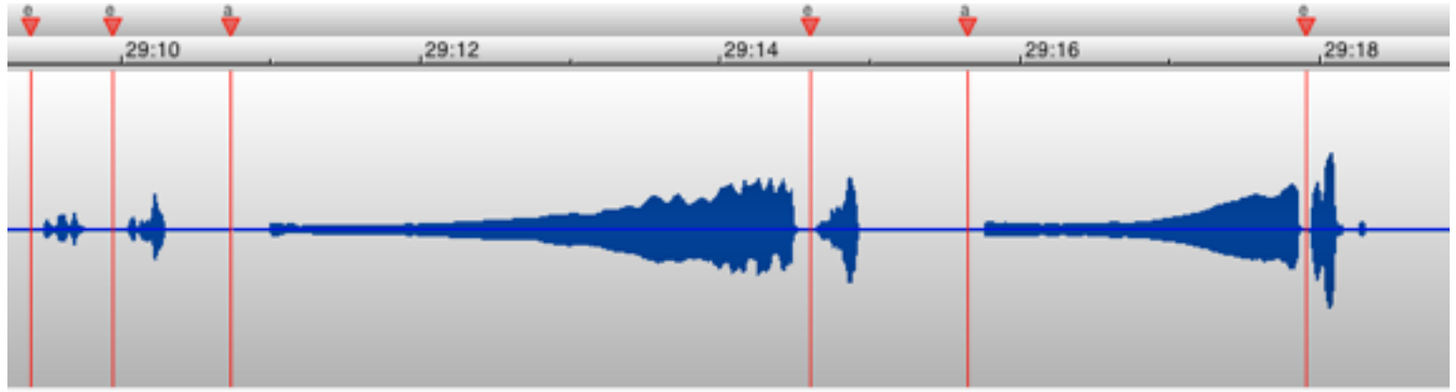
p peak_limiter

p compressor

Résidence Marta Gentilucci
(collaboration ISMM et AnaSynth)

z	Ribattuto disto long
y	Ribattuto disto short
x	Articulated vowels long
w	Articulated vowels short
v	Glottis' staccato
u	Grumble disto low long
t	Grumble disto low short
s	Grumble disto high long
r	Grumble disto high short
q	Grumble disto med long
p	Grumble disto med short
o	Distortion long
n	Distortion short
f	silence
a	tenuto





Scenarios

a a a e e e e a a a b b

a e e e e e e e e a f a e e e e e e e e e a a a b b b

a f e f e f f e e e f f a b b f b b a e e f e e e a

r r r r r s r r r r r s s u r r r u u t t t t u v v v v v v v v t t t u v v v v v v v v v v

a x w x x w x x o w w o o n n n n n t t t t t v v v v v v

o o o w w z y y z w w y y z z w w u t t t t t u t t t t t

Rodolphe Burger - Album « G.O.O.D »

Track generated using the memory…

Keyboard track from song 1

Ebow track from song 2

Guitar_1 track from song 3

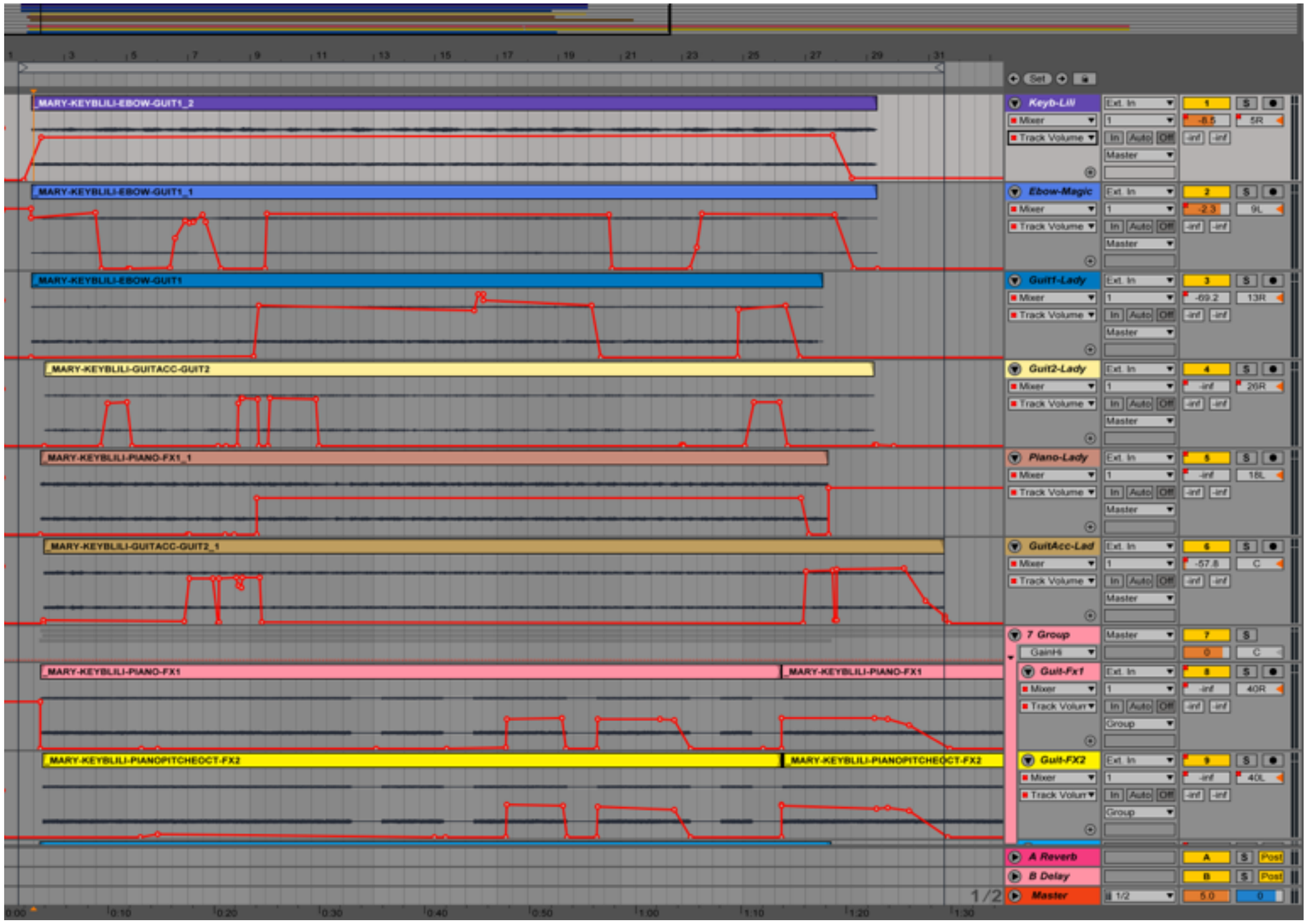
Guitar_2 track from song 3

Piano track from song 4

Guitar track from song 4

Guitar_1 track from song 5

Guitar_2 track from song 5



G. Bloch « La meute » / vidéo

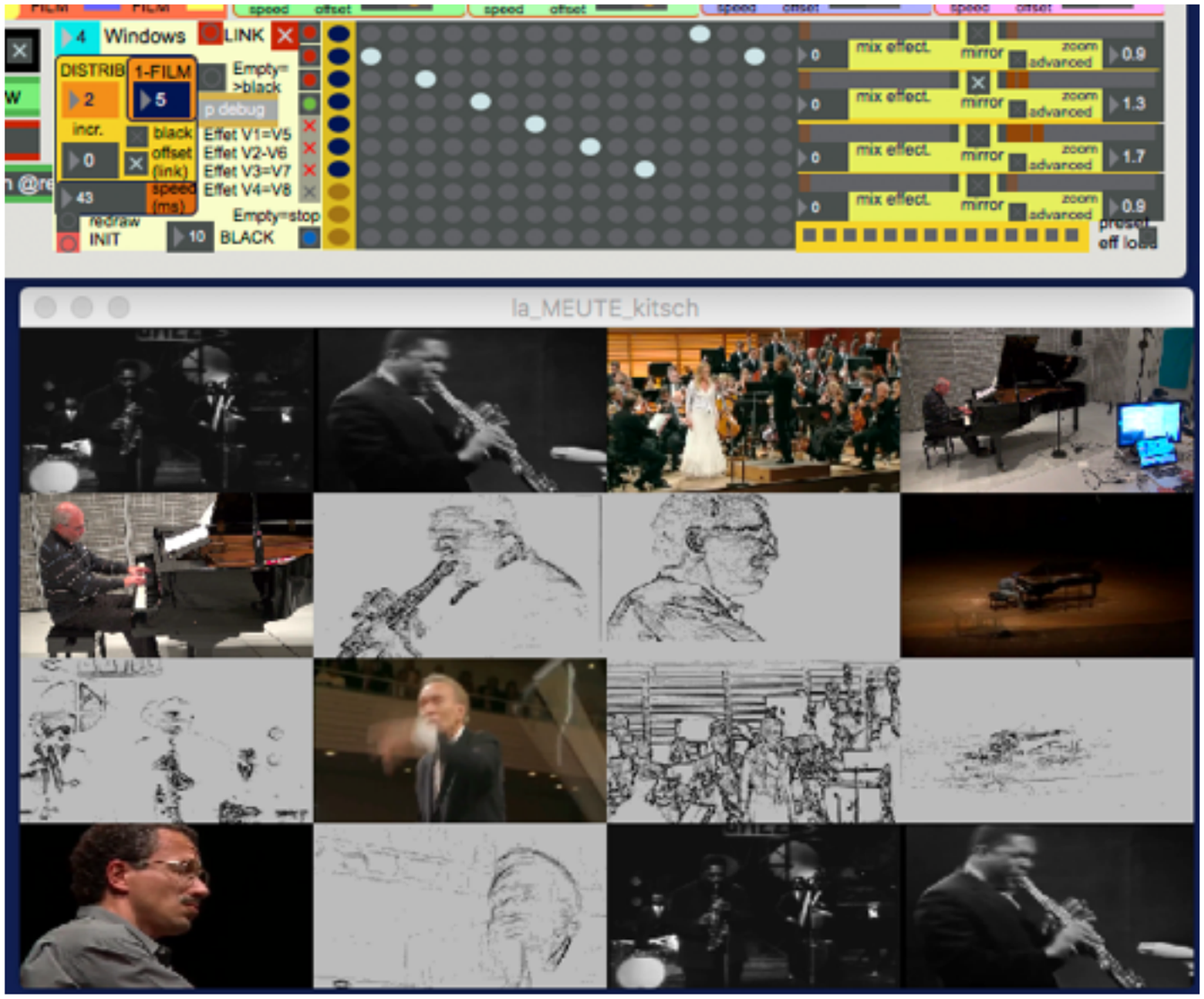


Schéma architecture commune DYCI2 - ImproteK - Somax
Version « simplifiée »

Utilisateur via interface
et / ou
Jeu du musicien live via module d'écoute réactive

Flux de requêtes non ordonnées

Index du premier événement d'improvisation à générer
float
Liste de labels / « poignées » requis(es)
liste de symboles ou vecteurs
Durée relative
float

Handler
Demand-driven

Connaissance de l'état courant du processus génératif
Trace d'execution

2

Modèles génératifs

1

Flux de fragments impro symbolique générés

Index événement dans l'improvisation
float
Index événement à jouer dans la mémoire
float
Liste de transformations à appliquer
liste de couples (identif. transfo : symbole , valeur : float)

Flux mesure du temps de la performance

Ordonnanceur / séquenceur
Time-driven, event-driven

Connaissance du temps courant de la performance

lecture dates associées à évènement

Segmentation mémoire

Table : index évènement dans la mémoire —> dates associées
Clé : float —> Valeur : Liste(date_ms : float , date_ms_float)

3

Analyse de la mémoire

Mémoire : description symbolique

Mapping symbolique
Table index évènement —> liste de labels - « poignées »

Mémoire : contenu segmenté

Segmentation
Table index évènement —> dates en ms
Pointeur vers contenu musical
i.e. vers fichier ou flux audio ou midi

Mémoire musicale (offline ou online)

Flux de contrôle moteur de restitution

Dates à lire
(float , float)
Coeff time stretch
float
Transformation du contenu
(symbole, float)

Moteur de restitution adaptatif

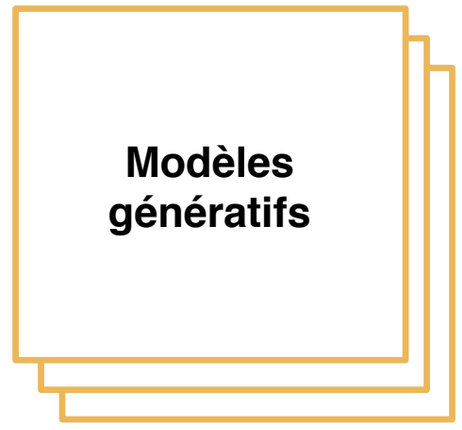
4

Également en charge des cross-fades aux discontinuités, des traitements et transformations.

Modèles génératifs

- Modèles génératifs : librairies.

- Modulaire, permettant utilisations autonomes (et modifications, expérimentations offline, etc…).



Modèles génératifs

- Classe **Model**
  - Séquence
  - Séquence de labels
  - Meta-données (i.e. informations données par le modèle)
- Classe **ModelNavigator**
  - méthode free_generation
  - méthode simply_guided_generation
  - méthode scenario_guided_geneation
  - paramètres de contrôle
- Module string matching

TODO 1

(Agent Somax Python/Max)
ou
(Agent ImproteK réactif+
Python/Max)

Faire rentrer code Somax dans ce schéma →

Agent
(Somax ou ImproteK réactif+)
Python/Max

Schéma architecture commune DYCI2 - ImproteK - Somax
Version « simplifiée »

Utilisateur via interface
et / ou
Jeu du musicien live via module d'écoute réactive

Flux de requêtes non ordonnées

Index du premier événement d'improvisation à générer
float
Liste de labels / « poignées » requis(es)
liste de symboles ou vecteurs
Durée relative
float

Handler
Demand-driven

Connaissance de l'état courant du processus génératif
Trace d'execution

2

Modèles génératifs

1

Flux de fragments impro symbolique générés

Index événement dans l'improvisation
float
Index événement à jouer dans la mémoire
float
Liste de transformations à appliquer
liste de couples (identif. transfo : symbole , valeur : float)

Flux mesure du temps de la performance

Ordonnanceur / séquenceur
Time-driven, event-driven

Connaissance du temps courant de la performance

lecture dates associées à évènement

Segmentation mémoire

Table : index évènement dans la mémoire —> dates associées
Clé : float —> Valeur : Liste(date_ms : float , date_ms_float)

3

Analyse de la mémoire

Mémoire : description symbolique

Mapping symbolique
Table index évènement —> liste de labels - « poignées »

Mémoire : contenu segmenté

Segmentation
Table index évènement —> dates en ms
Pointeur vers contenu musical
i.e. vers fichier ou flux audio ou midi

Mémoire musicale (offline ou online)

Flux de contrôle moteur de restitution

Dates à lire
(float , float)
Coeff time stretch
float
Transformation du contenu
(symbole, float)

Moteur de restitution adaptatif

4

Également en charge des cross-fades aux discontinuités, des traitements et transformations.

2 *Handler* : agent de génération demand-driven

- **Description** :
  Agent « demand-driven » qui embarque les modèles génératifs et leurs appels, et une description symbolique de la mémoire (online ou offline) utilisée pour la génération.
  Génère des fragments musicaux à la demande.
  Connaît l'état courant des processus génératifs pour assurer la cohérence entre les requêtes successives et les potentielles réécritures.

- **Entrées**

  - Données - mémoire

  - Flux de requêtes :

    - Exemples « à la ImproteK » :

      - *génération correspondant à* la grille d'Autumn Leaves
        à partir du temps 0 de l'improvisation
        pour toute la grille

      - *génération correspondant à* Lam7 / Dm7 / G7 / CMaj7
        à partir de la mesure 37
        pour 4 mesures

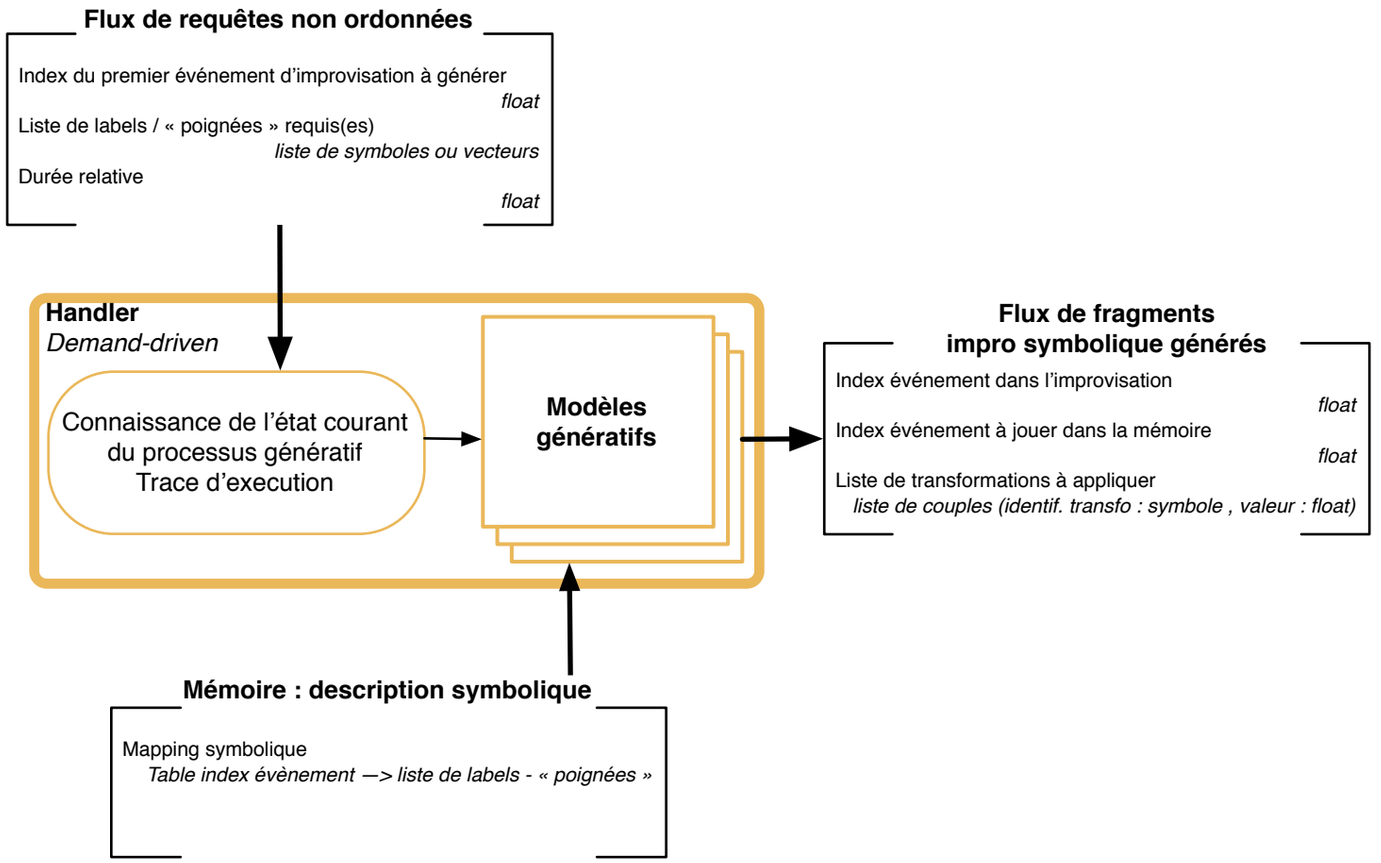
    - Exemple « à la Somax »

      - *génération correspondant* au chroma (0.67 ; 0,23 … ; 0,12)
        à partir de maintenant
        pour un évènement (saut) puis recopier la mémoire

    - Exemple « DYCI2 » : les deux, avec priorité sur l'un ou sur l'autre, et l'intermédiaire : scénario court-terme provenant d'un module d'écoute / inférence proposant une « continuation » de la structure sous-jacente déduite de ce que joue le musicien.

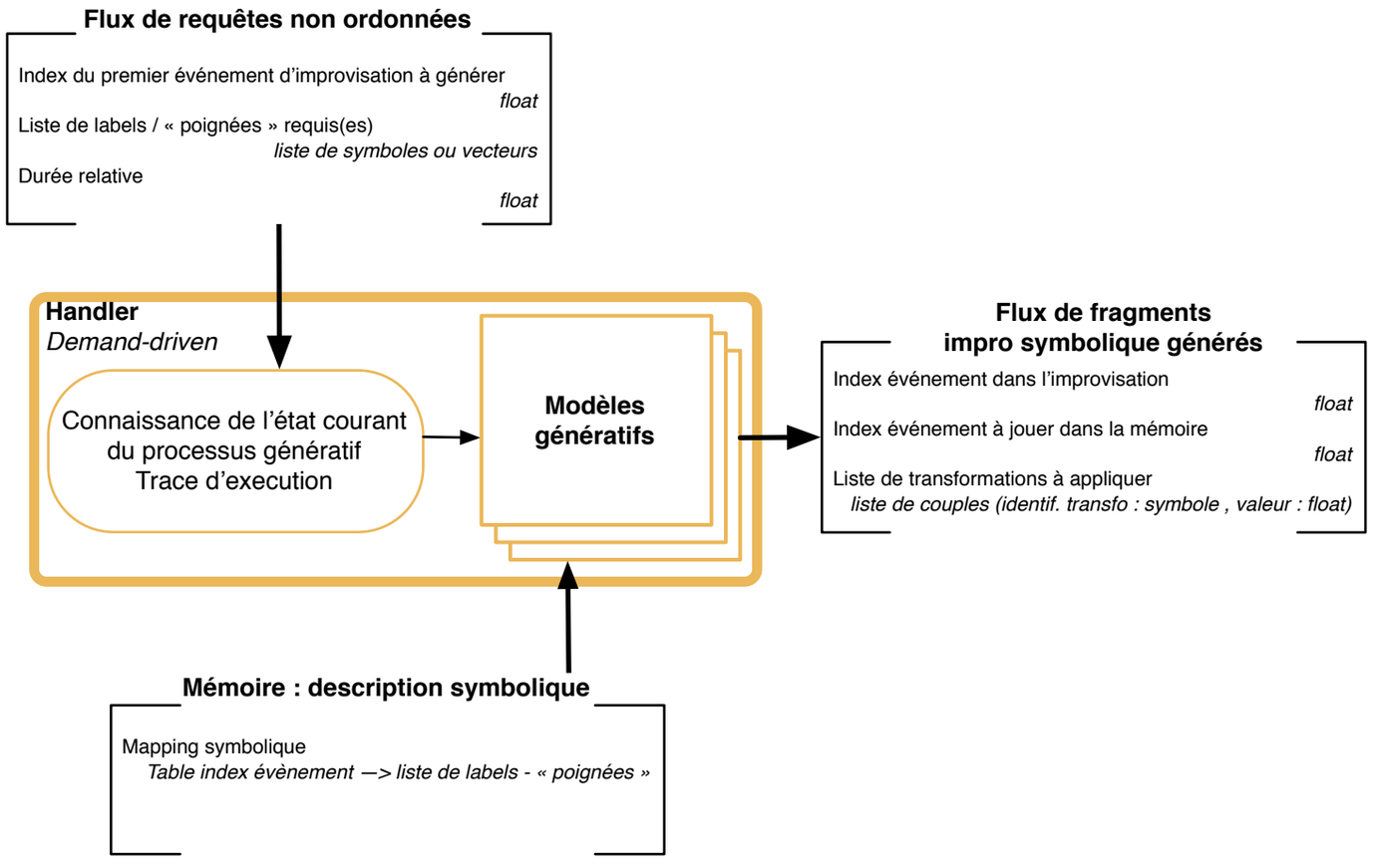
- **Sorties**

  - Fragments d'improvisation symbolique correspondant à la requête.



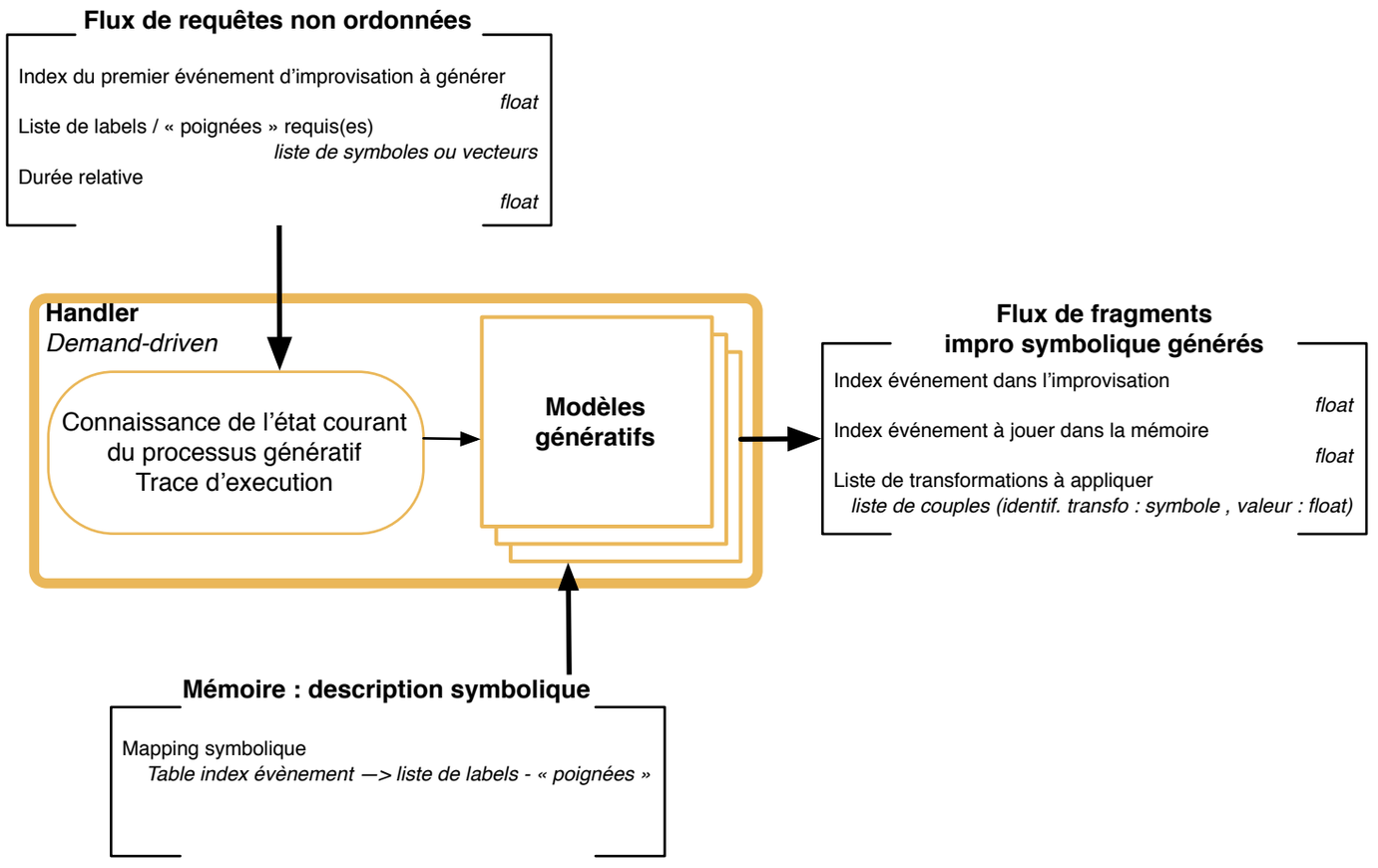
2 *Handler* : agent de génération demand-driven

- **Comparaisons** :
  - Classes :
    - Dans ImproteK correspondrait à la classe *Tune* ou à la « nouvelle » classe *Improvisation Handler* (équivalent de *Tune* mais adaptée pour le temps-réel, gérant les requêtes, etc…).
    - Dans Somax à rapprocher de ce qu'Axel appelle un « atome » ? (entité modèle génératif + mémoire + profils) ?
  - Connaissance de l'état courant du processus génératif:
    - Dans ImproteK : valeur courante de la continuité, dernière position de la tête de lecture dans l'oracle, valeur courante de l'ensemble des paramètres secondaires (transpos autorisées, contraintes sur longueur des préfixes…), trace d'execution pour pour pouvoir réécrire sur partie du scénario déjà généré de manière cohérente…
    - Dans Somax : valeur courante des paramètres de génération, en premier lieu le(s) profil(s) d'activité.
- **Gestion des requêtes** :
  - Dans une utilisation offline, « temps réel différé », ou sur une seule dimension / échelle temporelle, les requêtes peuvent être relativement peu fréquentes. « Flux de requêtes non ordonnées » : l'idée bonus est que cet agent puisse aussi, pendant une performance, recevoir pêle-mêle des requêtes hétérogènes avec valeurs et time-stamps (pour la date d'application de la requête dans le temps de l'impro) :
    - venant de l'utilisateur ou d'une écoute réactive ;
    - portant sur des paramètres de « guidage déclaratif» (valeur de descripteur requise, portion de scénario…) ou sur des paramètres secondaires plus bas-niveau (changement de paramètres des modèles) ou sur des échelles temporelles différentes ;
    - s'appliquant au moment où la requête est émise (« maintenant réagir à ce chroma », « maintenant telle grille ») ou avec un délai (« finalement avant de passer à la partie B de la grille, je pense qu'on refera un tour de A », « dans 4 temps favoriser les tranches dans un registre aigu »…)
  - Il gère les requêtes concurrentes, les requêtes en attente, les merge, les concatène, les ordonne, les met à jour… et s'occupe de lancer les bonnes executions des modèles génératifs en temps et en heure en gérant les accès concurrents à la mémoire.
  - (Architecture déjà élaborée, et prototype développé en Lisp : voir chapitre 9 de ma thèse.)



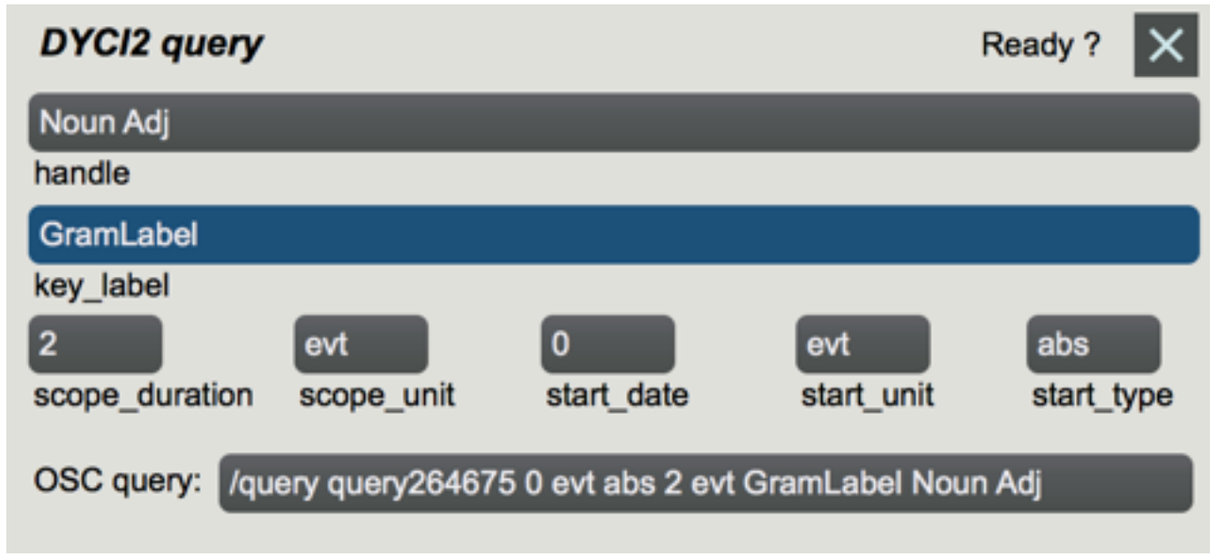
2 *Handler* : agent de génération demand-driven

- **Résumé :**
  - agent avec des entrées et sorties simples, générant des fragments d'impros symboliques à la demande avec une granularité temporelle spécifiée (« la réponse à cet évènement tout de suite à la Somax » , « sur tout le scénario » à la ImproteK offline, « aussi loin que tu peux » phase de génération / anticipation à la « ImproteK réactif »).
  - « Demand-driven » et asynchrone : est au courant de l'état courant des processus génératifs mais n'a pas besoin d'être connecté au temps courant de la performance.
- **Avantages** :
  - Ainsi peut être utilisé comme agent réactif pendant une performance, être intégré à système qui comprend ordonnanceur / séquenceur (Max) et recevoir ses requêtes de l'environnement.
- Peut également être utilisé offline comme interface, exemple :
  - Utilisation offline «100% à la ImproteK » : générer une séquence musicale avec scénario et mémoire donnés.
  - Utilisation offline «100% à la Somax » : en dehors de générer un accompagnement pour un fichier son de sax solo par exemple, on peut imaginer une interface où on déterminerait certains « hotspots » sur une grille temporelle afin de générer une séquence passant par ces points.
- Permettrait donc à terme de faire également un module pouvant être intégré une fois compilé dans d'autres environnements ayant leurs propres moteurs de restitution et systèmes d'ordonnancement (Live, OpenMusic…).
  Doit être écrit dans le même langage que modèles génératifs ou dans un langage qui permette de compiler le tout (i.e. pas dans Max).



2 *Handler* : agent de génération demand-driven

- Classe **Query**



- Classe **Generator**
  - Slot MemoryModel
  - Slot Output
  - Method process_query

- Classe **GenerationHandler**
  - Hérite de Generator
  - Slot performance_time
  - Slot LAST output
  - Slot CURRENT WHOLE output
  - Slot EXECUTION TRACE
  - Method process_query surchargée

2 *Handler* : agent de génération demand-driven

Classe Generator

```
    def process_query(self,query,print_info = False):
        """ Documentation."""
        self.current_generation_query = query
        self.generation_matching_query(query = self.current_generation_query,  print_info = print_info)
        self.current_generation_query.status="processed"
```

Classe GenerationHandler

```
#TODO ON EN EST LA
def process_query(self,query,print_info = False):

    if (self.current_performance_time["event"] >= 0 and self.current_performance_time["ms"] >= 0) or query.start["type"] == "absolute":
        if query.start["type"] == "relative":
            query.relative_to_absolute(current_performance_time_event = self.current_performance_time["event"],
                current_performance_time_ms = self.current_performance_time["ms"])


    if query.start["unit"] == "event":
        index_for_generation = query.start["date"]
        if index_for_generation <= self.current_generation_time["event"]:
            self.go_to_anterior_state_using_execution_trace(index_for_generation - 1)

    elif query.start["unit"] == "ms":
        if query.date["ms"] >=  self.current_generation_time["ms"]:
            index_for_generation = len(self.generation_trace)
        else:
            index_for_generation = self.index_previously_generated_event_date_ms(query.date["ms"])
            self.go_to_anterior_state_using_execution_trace(index_for_generation - 1)

    Generator.process_query(self,query, print_info = print_info)

    l = 0
    if not self.current_generation_output is None:
        l = len(self.current_generation_output)

    if query.start["unit"] == "event":

        if index_for_generation > len(self.generation_trace):
            for i in range(len(self.generation_trace),index_for_generation):
```

2 *Handler* : agent de génération demand-driven

TODO 2

Agent
(Somax **ou** ImproteK réactif+)
Python/Max

Processing pool of concurrent queries

+ redirection of outputs

Agent
(Somax **et** ImproteK réactif+)
Python/Max

2 *Handler* : agent de génération demand-driven

TODO 2

Agent
(Somax **ou** ImproteK réactif+)
Python/Max

Processing pool of concurrent queries

+ redirection of outputs

Agent
(Somax **et** ImproteK réactif+)
Python/Max

New incoming query

« Process »
=
« Run »

Output
=
events in
the memory

Send to sequencer / renderer

« Process »
=
« Insert in a query pool »

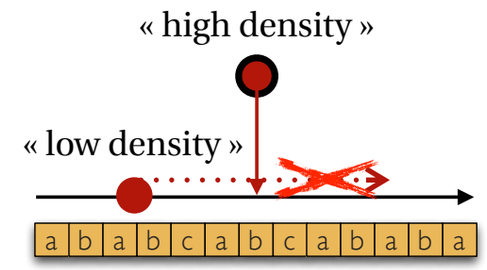
New incoming query

2 *Handler* : agent de génération demand-driven

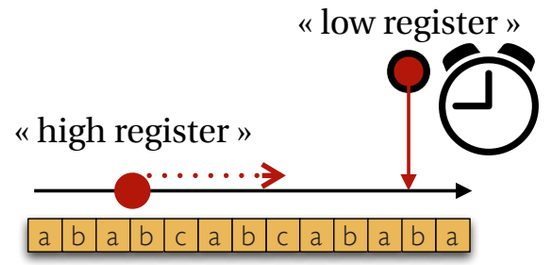
ALGORITHM 2 . Concurrent runs and new incoming queries

```
1  Q_i, query for improvisation time q_i
2  RQ, set of currently running or waiting queries
3  CurPos(Q), current generation index of Run(Q)
4  Whenever RQ = {Q} and Q not running do
5  |  Run(Q)
6  Whenever new Q received do
7  |  for Q_i ∈ RQ do
8  |  |  if q = q_i then
9  |  |  |  if Q and Q_i from same inputs then
10 |  |  |  |  Kill(Q_i)
11 |  |  |  else
12 |  |  |  |  Merge Q and Q_i
13 |  |  |  end
14 |  |  else if q > q_i then
15 |  |  |  if q < CurPos(Q_i) then
16 |  |  |  |  Relay(Q_i,Q,q)
17 |  |  |  else
18 |  |  |  |  WaitForRelay(Q_i, Q, q)
19 |  |  |  end
20 |  |  else if q < q_i then
21 |  |  |  WaitForRelay(Q, Q_i, q_i)
22 |  |  end
23 |  end
```

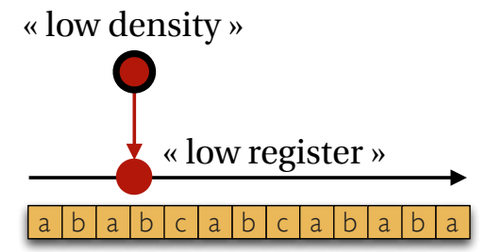
Relay



Wait for relay

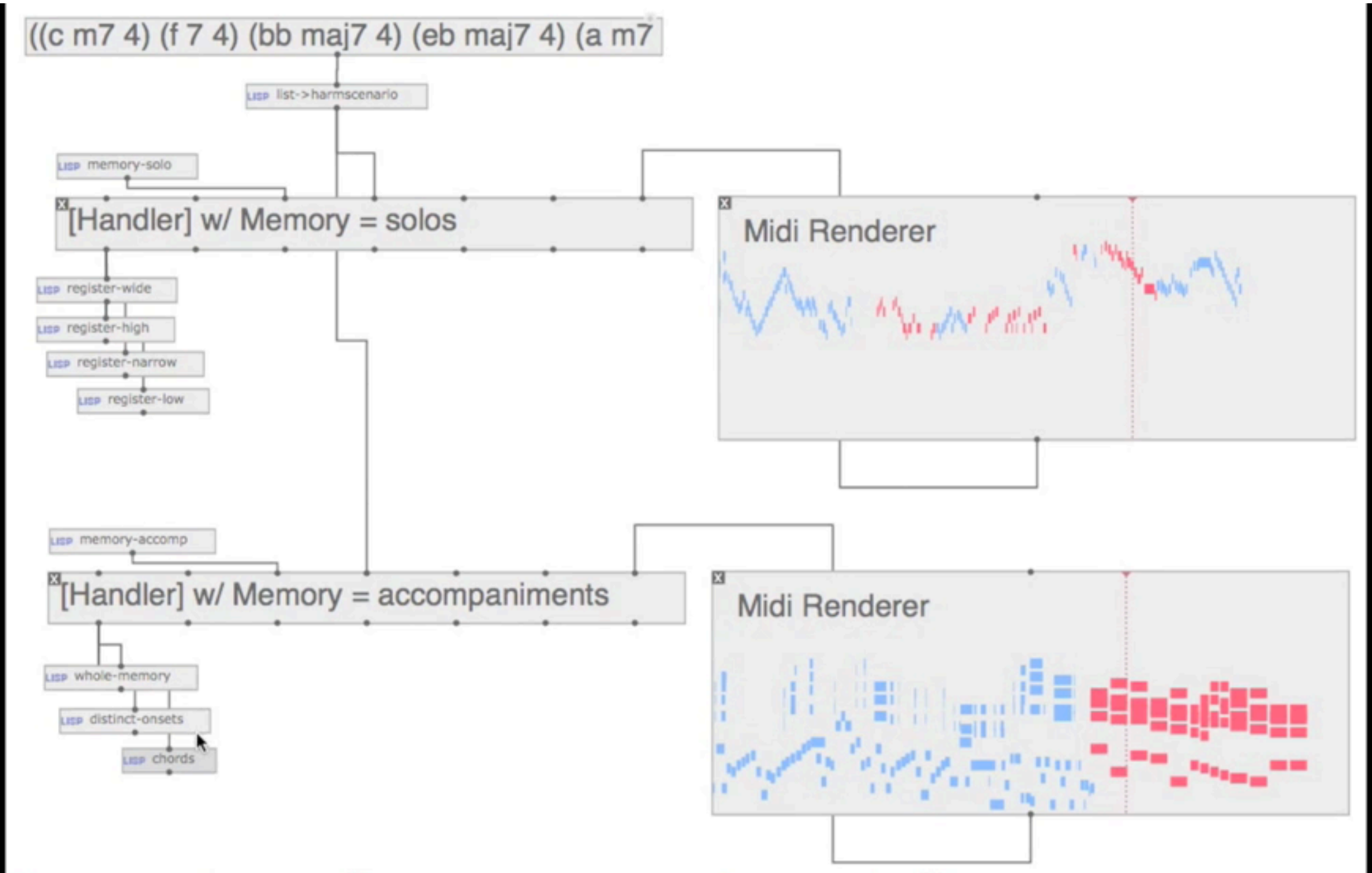


Merge



- Run(Q): start generation associated to Q. This function outputs generated data when it finishes,
- Kill(Q): stop run associated to Q and discard generated improvisation,
- Merge (Q_1, Q_2): create a new query [2] in which the list of impacted generation parameters and their associated new values correspond to the concatenation of that of Q_1 and Q_2,
- Relay(Q_1,Q_2,q): output the result of Q_1 for $[q_1; q[$, kill Q_1 and run Q_2 from q. The execution trace is read to maintain coherence at relay time q,
- WaitForRelay(Q_1,Q_2,q): Q_2 waits until Q_1 generates improvisation[3] at time q. Then Relay(Q_1,Q_2,q).

2 *Handler* : agent de génération demand-driven



Reaction: "accomp: chords"

2 *Handler* : agent de génération demand-driven

TODO 2

Agent
(Somax **ou** ImproteK réactif+)
Python/Max

Processing pool of concurrent queries

+ redirection of outputs

Agent
(Somax **et** ImproteK réactif+)
Python/Max

Query → *Run* → Output = events in the memory → *Send to sequencer / renderer*

Query → *Run* → Output = events in the memory → Modification current state of the model / filtering anticipations → Output = events in the memory → *Send to sequencer / renderer*

Schéma architecture commune DYCI2 - ImproteK - Somax
Version « moins simplifiée »

Metronome

Utilisateur via interface
et / ou
Jeu du musicien live via module d'écoute réactive

Flux mesure du temps de la performance

Flux de requêtes non ordonnées

Index du premier événement d'improvisation à générer — *float*

Liste de labels / « poignées » requis(es) — *liste de symboles ou vecteurs*

Durée relative — *float*

Ordonnanceur / séquenceur
Time-driven, event-driven

Connaissance du temps courant de la performance

lecture

Buffer anticipations d'impro générées par le modèle

Table : index évènement dans l'improvisation —> index à jouer dans la mémoire et transformations associées

Clé : float —> Valeur : Liste(float,liste(symbole, float))

index evt > now
écriture

index evt = now
lecture

lecture dates associées à évènement

adaptation tempo

Segmentation mémoire

Table : index évènement dans la mémoire —> dates associées

Clé : float —> Valeur : Liste(date_ms : float , date_ms_float)

Handler
Demand-driven

Connaissance de l'état courant du processus génératif
Trace d'execution

Modèles génératifs

Flux de fragments impro symbolique générés

Index événement dans l'improvisation — *float*

Index événement à jouer dans la mémoire — *float*

Liste de transformations à appliquer
liste de couples (identif. transfo : symbole , valeur : float)

Analyse de la mémoire

Mémoire : description symbolique

Mapping symbolique
Table index évènement —> liste de labels - « poignées »

Mémoire : contenu segmenté

Segmentation
Table index évènement —> dates en ms

Pointeur vers contenu musical
i.e. vers fichier ou flux audio ou midi

Mémoire musicale (offline ou online)

Flux de contrôle moteur de restitution

Dates à lire — *(float , float)*

Coeff time stretch — *float*

Transformation du contenu — *(symbole, float)*

Moteur de restitution adaptatif

Également en charge des cross-fades aux discontinuités, des traitements et transformations.

DYCI2 *Agent*

- Classe OSC Server
- Classe OSC Agent
- —> Démo patch Max