Projet DYCI2,
WP4 Intégration, expérimentation, validation, et retour d'usage
SP4.2 Intégration suite logicielle

Rapport de livrable :

L4.2 Intégration suite logicielle

Livrable	Date	Contributeurs	Rédacteurs	Contenu
L4.2	Août 2018	J. Nika (Univ. la Rochelle / Ircam-STMS), Ken Déguernel (Univ. la Rochelle / Ircam-STMS), A. Chemla-Romeu-Santos (Univ. Milan / Ircam-STMS), J. Bresson (Ircam-STMS), V. Siguret (ENS Lyon/ Ircam-STMS)	J. Nika, K. Déguernel, J. Bresson	Maquette Logicielle Agent DYCI2 0.3, Rapport scientifique

Résumé
La suite logicielle présentée dans ce rapport enrichit les agents improvisateurs de la dernière technologie L3.3.2 issue du WP3 (qui assemblait un continuum de paradigmes de générations allant de la planification à la réactivité) de modules probabilistes pour la génération et de modules d'écoute structurante et prédictive issus des recherches WP1 et WP2. Cette troisième architecture s'accompagne de l'encapsulation et de l'intégration de technologies issues du projet DYCI2 dans des environnements externes dédiés à la performance et à la composition musicale.

Répertoire de développement logiciel collaboratif
https://forge.ircam.fr/p/DYCI2_library/
https://github.com/DYCI2/om-dyci2/

Adresse du livrable logiciel
DYCI2_WP4.2_L.4.2.zip
sur
https://forge.ircam.fr/p/Dyci2/

La suite logicielle présentée dans ce rapport enrichit les agents improvisateurs de la dernière technologie L3.3.2 issue du WP3 (qui assemblait un continuum de paradigmes de générations allant de la planification à la réactivé) de modules probabilistes pour la génération et de modules d'écoute structurante et prédictive issus des recherches WP1 et WP2.
La finalité est double : proposer une librairie de processus génératifs pouvant être interfacés avec d'autres environnements (par exemple via le protocole OSC) ainsi qu'une librairie d'agents musicaux (pouvant générer des séquences audio ou midi, de manière autonome ou guidée, en faisant appel à la librairie de processus génératifs). La « librairie DYCI2 » est donc constituée de deux librairies : une librairie Python implémentant les processus génératifs (modèles de mémoire et traitement des requêtes) et architectures réactives dans le domaine symbolique, et une librairie d'objets Max pour la performance proposant des interfaces pour le choix de mémoires musicales et de paramètres pour leur apprentissage, l'envoi de requêtes pour guider la génération, et des modules de restitution pour jouer les séquences générées dynamiquement.
Cette troisième architecture s'accompagne enfin de l'encapsulation et de l'intégration de technologies issues du projet DYCI2 dans des environnements externes dédiés à la performance ou la composition musicale.

Agent DYCI2

Les différentes stratégies de génération d'un agent improvisateur embarquant une « mémoire » musicale ont été étudiées et développées en parallèle au sein des différentes sous-tâches du WP3, puis assemblées dans l'intégration décrite par le rapport L3.3.2. Nous invitons le lecteur à se reporter à ce document et rappelons simplement ici les différents paradigmes pouvant être mobilisés par un agent DYCI2 au sein d'une même performance : de la génération libre à la génération structurée par un scénario temporel, en passant par la réaction aux évènements extérieurs analysés par un module d'écoute.

Génération libre. Une fois le modèle de mémoire construit, une requête « free » entraîne la génération d'une séquence musicale inédite suivant la logique temporelle interne du matériau appris, sans autre contrainte de structure ou d'écoute réactive.
Guidage par scénario long-terme. La génération peut être guidée par un « scénario » déterminé par l'utilisateur. Cette séquence symbolique de référence est définie sur le même alphabet que les annotations de la mémoire musicale utilisée (labels d'accords, modes de jeu, etc.) et guide l'improvisation (séquence d'accords, séquence de modes de jeu, etc.). Il s'agit donc de trouver des segments de la mémoire correspondant aux portions successives du scenario à suivre et de les enchainer de manière créative. Pour atteindre cet objectif, le modèle proposé associe à chaque instant de la génération l'*anticipation* en assurant la continuité avec le futur du scénario, et la *cohérence avec la logique musicale de la mémoire* en assurant la continuité avec le passé de la mémoire (voir ImproteK, L3.1.1, pour plus de détails).
Guidage par scénarios dynamiques à court terme. L'implémentation des stratégies de gestion de requêtes concurrentes et de l'approche de la réaction en tant que réécriture d'anticipations préalablement générées (exposées dans L3.1.1) a permis d'introduire dans L3.3.1 la notion de scénarios dynamiques à court terme. Au cours de la performance, un opérateur-musicien peut ainsi envoyer des requêtes correspondant à des séquences de labels spécifiant ce que l'agent doit générer et jouer sur le champ, tout en maintenant la cohérence avec ce qui vient d'être joué ainsi qu'avec les anticipations préalablement générées dans le cas où une révision des anticipations est nécessaire. Ce paradigme de guidage introduit un mode de jeu que l'on pourrait qualifier de « meta DJing » ou « DJing d'intentions » : en effet, un opérateur-musicien peut improviser en contrôlant un agent à l'échelle de la narration musicale (par exemple « à partir du temps prochain : générer et jouer une séquence correspondant à la suite d'accords Dm7 G7 CMaj7», « maintenant : générer et jouer une séquence partant de 'grave rugueux' pour arriver 'aigu brillant' »).

Génération libre structurée. La modularité de la librairie permet de chaîner deux (ou plusieurs) agents de manière à ce que le type de contenus retourné par le premier soit le type de labels utilisé pour construire les requêtes pilotant le second. Ce type d'utilisation permet d'ajouter une dimension verticale à l'improvisation avec, par exemple, une chaîne dans lequel un agent se spécialise dans l'harmonisation et le second dans l'arrangement. Il permet également d'introduire un intermédiaire entre les stratégies de génération « libre » et « structurée ». Un premier agent peut ainsi, par exemple, naviguer de manière « libre » dans sa mémoire et retourner non pas les contenus musicaux des états par lesquels il passe, mais les labels associés. La séquence de labels ainsi construite sera donc inédite mais cohérente avec la structure de sa mémoire et pourra servir de scénario pour un autre agent générant du contenu musical.

Guidage par scénario de descripteurs audio. Les stratégies de guidage « par scénario » ou « par scénarios dynamiques à court terme » peuvent également être utilisées pour naviguer dans une mémoire constituée par l'analyse automatique d'un fichier ou d'un flux audio. Au cours de la performance, l'utilisateur peut ainsi envoyer des requêtes spécifiant des séquences de classes que l'agent doit générer et jouer. Les classes constituant l'alphabet de labels sont obtenues par une analyse du fichier « mémoire » selon une sélection de descripteurs effectuée par l'utilisateur, puis par un clustering discrétisant l'espace de descripteurs en un nombre de classes également choisi par l'utilisateur.

Guidage par écoute réactive. La navigation dans une mémoire ainsi constituée par analyse automatique peut également être pilotée par un module d'écoute/analyse temps-réel. Un flux audio capté en temps réel – par exemple un musicien co-improvisant avec l'agent – est analysé selon la même sélection de descripteurs que la mémoire pour créer automatiquement des requêtes de génération.

Guidage hybride écoute réactive / scénarios. La fusion de cette dernière stratégie avec le paradigme de scénarios à court terme a également permis d'introduire de nouvelles stratégies de générations pilotées par l'écoute entre lesquelles un agent peut alterner au cours d'une même performance : le label déterminé par l'écoute peut être utilisé pour créer un scénario court terme dans lequel il est répété, dont il est le point de départ, ou encore dont il est la destination. Ces stratégies introduisent un contrôle hybride entre le musicien produisant le stimulus et l'opérateur-musicien « composant » en temps réel les modalités de la réactivité dans une dynamique de compagnonnage humain / humain-machine.

La **librairie Python DYCI2** implémente les modèles temporels, stratégies de navigations guidées, et architectures d'agents génératifs. Sa documentation est disponible à l'adresse suivante : http://repmus.ircam.fr/downloads/docs/DYCI2_library/. Elle a été conçue de manière modulaire afin de pouvoir être prise en main par des développeurs souhaitant utiliser les outils qu'elle implémente pour créer leurs propres agents improvisateurs, étendre simplement le panel de stratégies de génération, et intégrer ces outils dans un environnement externe, par exemple en utilisant le protocole OSC (ou l'encapsulation en langage C décrite plus loin). Dans cette optique, un fichier tutoriel est associé à chacune de ses classes.
La **librairie Max DYCI2** propose une interface pour chacune des classes de la librairie Python ainsi que différents modules de rendu audio. Elle communique en arrière-plan avec la librairie Python via le protocole OSC et permet de construire aisément son propre dispositif de performance en associant un ou plusieurs agents, modules de rendu, sources de requêtes, et stratégies de génération. Des tutoriels associés à la librairie Max ont été réalisés en collaboration avec le saxophoniste Rémi Fox et la compositrice Marta Gentilucci. Ces patchs d'exemples ont été adaptés de patchs de concerts réalisés pour des créations associées au projet DYCI2. Ces exemples sont accompagnés de matériaux musicaux originaux créés par des artistes partenaires du projet.
Les sections suivantes décrivent l'intégration des résultats des autres « work packages » dans ce cadre établi par le livrable L3.3.2.

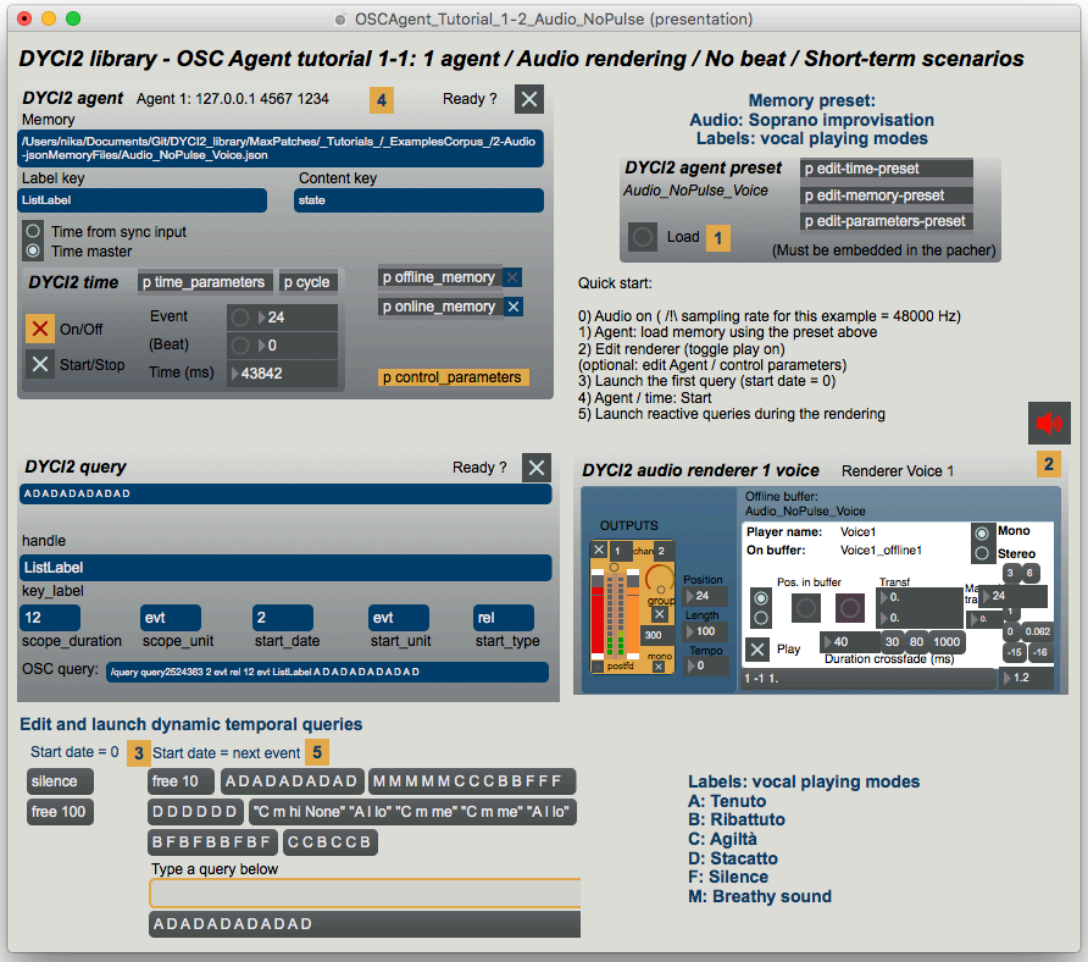


Figure 1. Exemple de patch tutoriel utilisant les objets de la librairie : 1 agent embarquant un modèle appris sur une mémoire audio annotée, des exemples de requêtes pouvant lui être envoyées pour guider la génération, 1 module de restitution audio permettant de jouer les séquences dynamiquement générées

(exemple issu de l'utilisation de la librairie DYCI2 par la compositrice Marta Gentilucci lors de sa résidence à l'Ircam, cf. livrable 4.1.4).

Modules probabilistes pour la génération

Cette section présente l'intégration dans la librairie Python DYCI2 des travaux effectués dans les « work packages » WP2.1 et WP2.3.

Représenter les connaissances extérieures par des modèles probabilistes

Pour l'intégration des méthodes de parcours d'un oracle des facteurs orienté par un modèle probabiliste permettant de prendre en compte les aspects multidimensionnels de la musique (WP2.1), nous avons introduit :
- une classe CondProb, héritant de la classe Modèle, pour représenter les différents sous-modèles probabiliste utiliser pour décrire les relations entre les différentes dimensions musicales analysés dans un corpus. Cela peut représenter des modèles probabilistes horizontaux comme des n-grammes ou des modèles verticaux pouvant décrire les relations entre plusieurs dimensions (par exemple, relations notes/accords).

- une classe ProbaNavigator, héritant de la classe Navigator pour définir la navigation d'une mémoire dirigée par des connaissances représentées par des modèles probabilistes ; la classe Model utilisé pour représenter la mémoire fournit les états atteignables de la séquence, ProbaNavigator effectue une interpolation des différents modèles probabilistes choisis par l'utilisateur pour définir des probabilités de transitions vers chacun de ces états, permettant un parcours de la mémoire dirigé par des connaissances multidimensionnelles (la représentation de la mémoire pouvant elle être construite en ne suivant qu'une dimension).

- une classe FactorOracleProbaNavigator, héritant de FactorOracle et de ProbaNavigator, utilisant la meta-classe MetaModelNavigator, pour décrire les méthodes de navigation d'un oracle des facteurs dirigé par un modèle probabiliste.

- un script FactorOracleProbaNavigator_tutorial.py proposant un exemple d'utilisation des classes présentées ci-dessus.

Introduire des scénarios temporels multi-niveaux

Pour l'intégration des méthodes de guidage de l'improvisation sur un scénario temporel multi-niveau (WP2.3), nous avons introduit :

- une classe MultiLevelNavigator, héritant de la classe Navigator décrivant les nouvelles heuristiques de parcours d'une mémoire permettant de respecter un scénario multi-niveau. Par exemple, une description d'une grille d'accord aux niveaux harmonique (accord), fonctionnel (groupe d'accords), et structurel (sections), constitue un scénario multi-niveau. L'importance des différents niveaux d'organisation décrit dans le scénario est définie par l'utilisateur. La navigation va choisir en priorité les zones mémoires respectant le plus de niveaux possibles avec le scénario, mais va aussi étendre les possibilités aux zones mémoires ne partageant que certains niveaux avec le scénario.

- une classe FactorOracleMultiLevelNavigator, héritant de FactorOracle et de MultiLevelNavigator, utilisant la méta-classe MetaModelNavigator pour définir les méthodes de navigation d'un oracle des facteurs respectant les contraintes d'un scénario multi-niveau.
- un script FactorOracleMultiLevelNavigator_tutorial.py proposant un exemple d'utilisation des classes présentées ci-dessus.

Modules d'écoute structurante et prédictive

Les recherches portant sur l'écoute structurante et prédictive du WP1 (L1.2) ont abouti au développement d'un prototype de module d'écoute et d'un prototype de module de prédiction prenant en entrée un flux audio segmenté par pulsation et retournant respectivement un label d'accord correspondant à la dernière pulsation obtenue, et à une progression d'accords inférée pour les mesures à venir.
Tout d'abord, le module d'écoute ouvre la voie à une nouvelle possibilité de constitution de mémoire musicale en temps réel. Celui-ci permettra en effet de construire en temps réel des mémoires annotées par un alphabet portant une sémantique musicale haut-niveau, les mémoires musicales apprises à partir d'un flux audio live étant jusqu'à ce stade annotées par des classes de descripteurs audios, renseignant donc plus sur le « signal » que sur la « musique » contenue dans chaque évènement.
L'association des stratégies de génération développées dans la première section et du module d'écoute prédictive - à l'état de prototype mais aux résultats prometteurs - introduit un nouvel intermédiaire entre les paradigmes de génération « guidée par un scénario à court-terme » et « guidée par une écoute réactive ». Les progressions d'accords à court terme émises par ce module peuvent constituer des requêtes de « scénario à court-terme » envoyées en temps réel à un agent, lui permettant ainsi de générer des improvisations anticipatives à partir d'un scénario dynamique inféré et non plus seulement prédéfini.

Intégration des technologies DYCI2 dans les environnements CatarRT et OpenMusic

La suite logicielle est également enrichie de l'encapsulation et de l'intégration de technologies issues du projet dans des environnements externes dédiés à la performance ou la composition musicale : CataRT et o7 (nouvelle implémentation de l'environnement OpenMusic).

CataRT
Un travail mené en collaboration avec Diemo Schwarz et l'équipe ISMM de l'Ircam a conduit au développement du nouveau module d'analyse/rendu utilisant l'environnement MuBu (http://forumnet.ircam.fr/fr/produit/mubu/). MuBu *multi-buffer* est un conteneur de données sonores et de mouvement fournissant de la mémoire structurée pour le son et le mouvement enregistrés à travers des interfaces et opérateurs en temps réel en tant qu'objets externes complémentaires pour Max. CataRT by MuBu (http://imtr.ircam.fr/imtr/CataRT), développé au sein de l'environnement Mubu, est un système de synthèse concaténative temps réel qui permet de jouer des « grains » sonores (à partir d'un grand corpus de sons segmentés et analysés par descripteur) selon la proximité avec une cible dans l'espace descripteur, au moyen d'une souris ou d'un contrôleur externe. Cette approche peut être vue comme une extension de la synthèse granulaire en donnant un accès à des caractéristiques sonores spécifiques. L'interaction repose sur une interface simple consistant en l'affichage d'une projection 2D de l'espace de descripteurs, et une navigation avec la souris, où les grains sont sélectionnés et joués par proximité géométrique

Figure 1. Exemple de patch tutoriel utilisant les objets de la librairie : 1 agent embarquant un modèle appris sur l'analyse automatique d'un fichier audio, un module de requêtes manuelles et requêtes provenant d'une écoute réactive d'un flux audio analysé en temps réel, un module de restitution audio basé sur le moteur CataRT (exemple issu de l'utilisation de la librairie DYCI2 par le saxophoniste Rémi Fox lors d'une performance à l'Académie de Darmstadt, juillet 2018).

Cette combinaison des recherches issues du projet DYCI2 et du moteur CataRT, décrite en détail dans le livrable L3.3.2, a permis la création de nouvelles stratégies de génération : le guidage par scénarios de descripteurs audios et le guidage par écoute réactive. Notons que cette nouvelle technologie enrichit CataRT d'une prise en compte de la temporalité du fichier « mémoire » et d'une optimisation des chemins choisis dans l'espace de descripteurs grâce à celle-ci.

OpenMusic

Un travail amorcé par le stage de Victoire Siguret, co-encadré par Jérôme Nika et Jean Bresson, a mené au développement d'une interface en langage C pour la librairie Python DYCI2 et de sa première application : l'intégration des agents DYCI2 dans l'environnement OpenMusic. OpenMusic (http://forumnet.ircam.fr/fr/produit/openmusic/) est un environnement permettant de développer des processus musicaux à l'aide de la programmation visuelle. L'utilisateur peut ainsi relier des modules fonctionnels entre eux et créer un "patch" afin de générer ou transformer une structure musicale. L'intégration des technologies DYCI2 a été plus précisément effectuée dans l'environnement o7 (https://openmusic-project.github.io/), plateforme destinée à devenir la prochaine version d'OpenMusic apportant de nouvelles fonctionnalités notamment en terme de programmation réactive, d'interface graphique, et de communication avec des environnements externes. De plus amples détails sont donnés dans la notice utilisateur d'*om-dyci2* ci-dessous.

Interface en langage C et intégration dans O7 (nouvelle version d'OpenMusic)

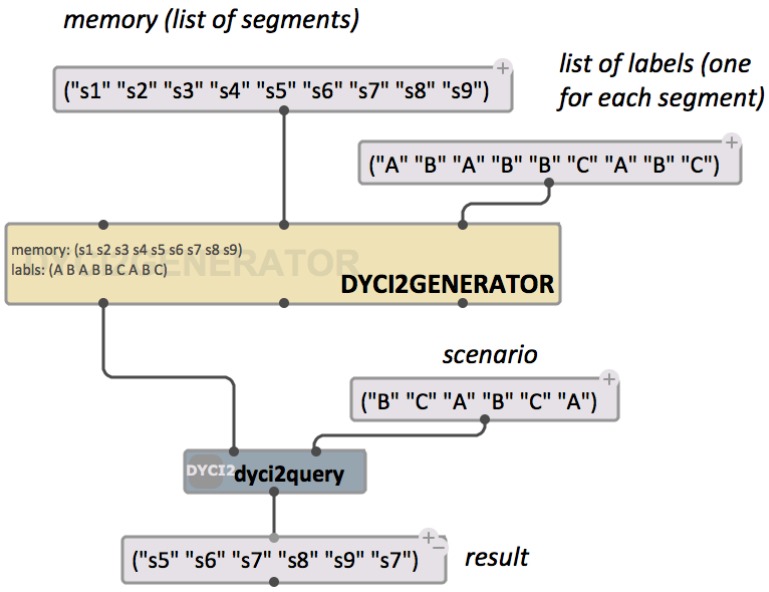
https://github.com/DYCI2/om-dyci2

The DYCI2 library was designed for improvisation and style imitation: it works on sequences of abstract labels (e.g. chord labels, etc.) matching segments of a memory. DYCI2 produces new sequences by recombinations of these memory segments, given a scenario expressed as a sequence of labels, and keeping some properties learned from the memory.

Basics

The main object in **om-dyci2** is `DYCI2GENERATOR`. It is initialized with a memory (list of "segments") and a sequence of labels indexing this memory (two lists of the same length). Currently `DYCI2GENERATOR` only support strings as memory segments and labels.

In the simple example below the memory is a simple list of 9 segments ("s1", ..., "s9") indexed by an alphabet of 3 labels ("A", "B", or "C"). Once the generator is initialized, a query can be made using the **dyi2query** function, and a new list of labels driving the sequence called the *scenario*.



The patch *dyci2generator-basic* provided with the library reproduces a similar example.

Musical sequences

In most case with real musical sequences, the memory will simply consist in indices or time markers indicating the beginning and end time of each segment. These markers can correspond to the markers in an audio file, chords or other events in a MIDI sequence, etc. (see examples below).

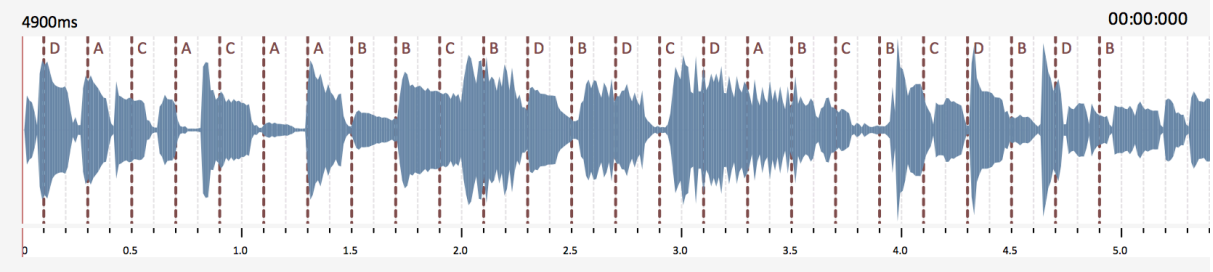
Depending on the application at hand, a first trick will be to convert both the list of time markers and the corresponding labels into lists of strings to initialize a DYCI2Generator. A second trick will be to recombine a sequence from the new list of memory segments.

Two cases are covered in the library examples:

- a segmented and labelled audio file
- a MIDI file including text labels.

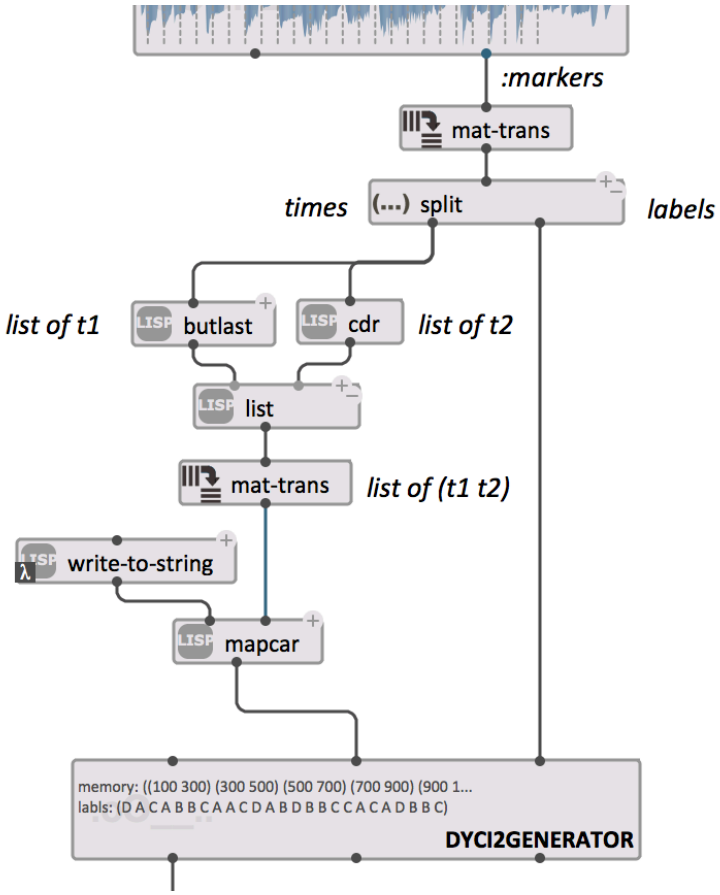
Audio file

Let's start with an OM audio file (SOUND object) containing labelled markers.



In o7, markers and labels can be set and extracted using the `:markers` inputs of the SOUND box. Markers can also be added manually (**CMD + click**) and labelled (select + **L**) within the SOUND object editor. Labeled markers are output as a list of ((time "label") (time "label") ...).
From this list, 2 objects need to be generated:

- A list of segments formatted as strings: ("(t1 t2)" "(t2 t3)" ...)
- A list of labels



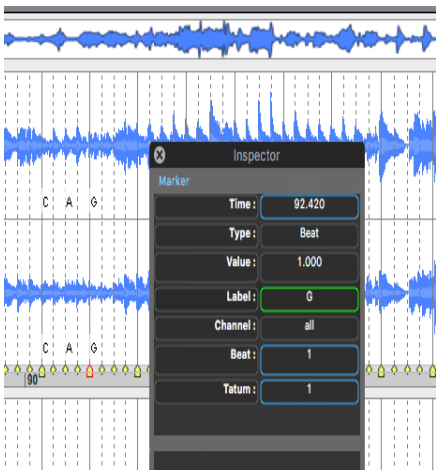
The result of a query is a new list of ("(ta ta+1)" ... "(tx tx+1)" ...). If the original material is an audio file, the segments corresponding to these dates can be retrieved and sequenced to be played. This can also be done using standard OM audio tools.

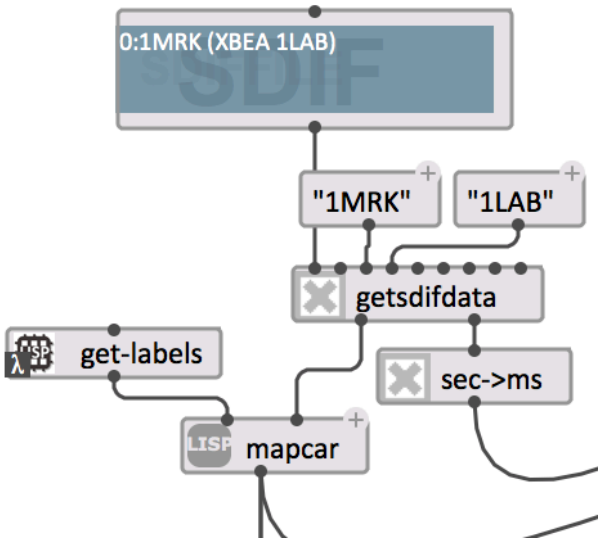
=> See the patch *dyci2generator-audio* provided in the library

Get markers and labels from an external software:

OM/o7 provide tools to process files and extract labels and time markers coming from external tools.

IRCAM's AudioSculpt for instance provides powerful means to automatically segment an audio file in beats and let you input labels attached to the beat markers. Markers can then be exported as SDIF files, and imported/processed in OM:





Depending on your annotation and storage convention, different patches will need to be programmed to retrieve the correct format for memory and labels.

MIDI

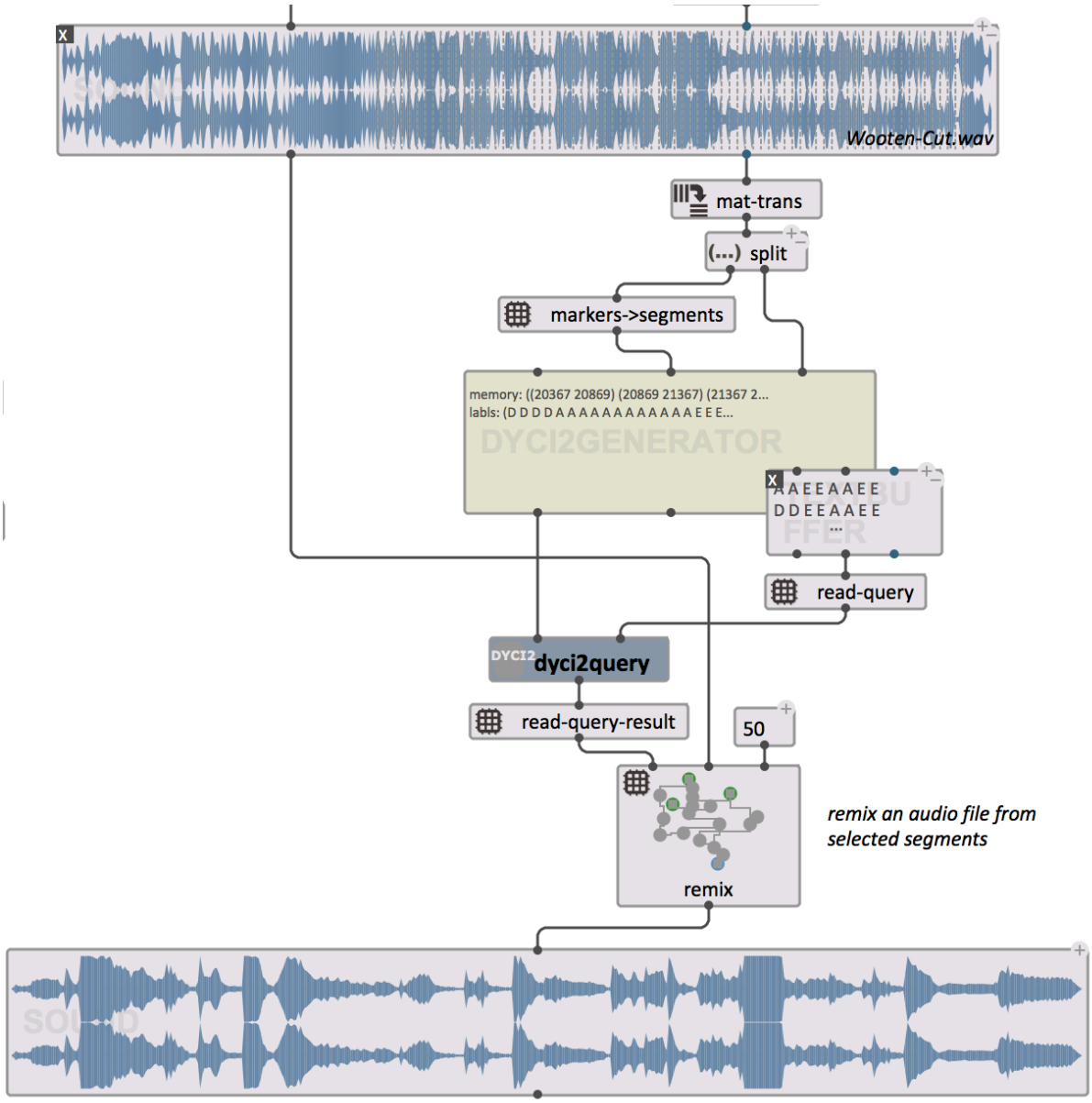
MIDI works similarly as in the previous audio examples, except that labels would most likely be encoded as MIDI messages (e.g. of type *Lyrics*). OM/o7 also provide tools to process and extract such messages, and use DYCI2 objects in a similar way.

Some examples are provided in the library.

Queries

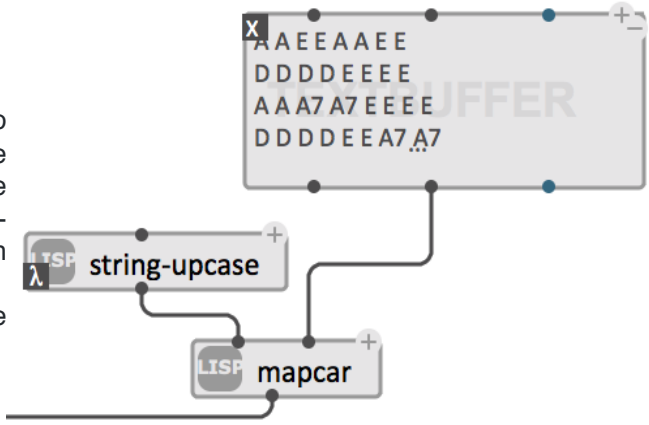
The function `dyci2query` allows to get a new generated sequence from a trained `DYCI2GENERATOR` object and new list of labels (or *scenario*).
Note: the labels in the scenario must all be part of the original training labels.

Depending on how the memory was formatted, adequate operations must be done to reconstitute a musical sequence (e.g. concatenating MIDI chucks, sequencing audio buffers, etc.)



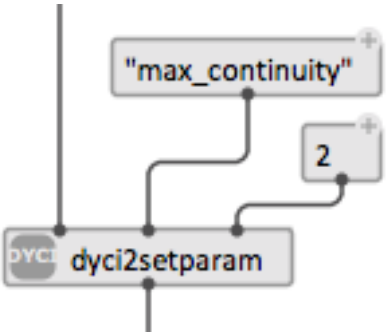
Query input

The TEXTBUFFER object can be convenient to easily input queries in dyci2query. Use the :read-mode input's option to get the adequate contents format (e.g. using the "flat-list" option), and convert to string to match the DYCI2GENERATOR labels' format.
=> Avoid space in labels, which might be wrongly parsed and processed.



Generation parameters

A lot of options and parameters in DYCI2 allow you to drive the scenario generation. The `dyci2setparam` command, allows to set parameters of the `DYCI2Generator`. Select the `parameter` (a string corresponding to its name — see DYCI2 documentation for more info) and set the `value`.



Two main parameters are currently used:

- `avoid_repetitions_mode`:
  - 0: avoids several use of the same segment from the memory in the output sequence.
  - 1: favors the least used segments for the output sequence.
  - 2: allows repetitions of segments in the output sequence.
- `max_continuity`: the maximum length of a segments sequence from the memory in the output sequence.