Domain Specific Language

Music ML

TeamB :

Yahiaoui Imène - Gazzeh Sourour

Ben Aissa Nadim - Al Achkar Badr

Sommaire

I. Introduction

Ce rapport présente le travail effectué pour mettre en œuvre un langage externe spécifique au domaine de la musique (External Domain-Specific Language, DSL externe), dénommé "MusicML", conçu pour les musiciens désireux de transcrire leurs compositions musicales en partitions. Ces partitions sont ensuite converties en artefacts compatibles avec le format MIDI, permettant leur lecture sur divers lecteurs MIDI.

Dans un premier temps, le rapport se consacre à l'exposition de la syntaxe abstraite élaborée pour modéliser le domaine musical concerné. Cette section aborde les décisions stratégiques prises pour intégrer les diverses caractéristiques musicales et explique comment les concepts clés du domaine ont été réifiés.

Poursuivant son analyse, le document se penche sur la syntaxe concrète, illustrée via la notation Backus-Naur Form (BNF). Cette partie met en lumière comment cette syntaxe a été conçue pour faciliter l'expérience utilisateur, en équilibrant la précision et l'expressivité tout en minimisant la charge cognitive imposée à l'utilisateur.

Le rapport présente ensuite une série de scénarios d'utilisation basiques, conçus pour démontrer les fonctionnalités fondamentales du DSL. Il aborde par la suite les extensions développées pour enrichir le DSL, permettant ainsi de prendre en charge des scénarios plus complexes et avancés grâce à des fonctionnalités supplémentaires.

Avant de conclure, une attention particulière est accordée à la présentation de l'add-on mis en place. Cette section vise à mettre en évidence ce service susceptible de stimuler l'adoption du DSL par les utilisateurs finaux.

Le document se clôt sur une synthèse critique, offrant une réflexion rétrospective sur les choix opérés au cours du développement du DSL. Cette conclusion aborde tant les réussites que les éventuelles améliorations ou erreurs, fournissant ainsi une évaluation équilibrée et instructive du projet.

Le dépôt github contenant notre projet est le suivant : si5-dsl-teamb.

II. Modèle du domaine

Notre modèle de domaine, basé sur la définition de MusicML du laboratoire et enrichi par des ressources externes, intègre des fonctionnalités clés et concrétise les concepts du MIDI pour une application pratique. Suivant est ce modèle qui est alimenté par notre DSL :

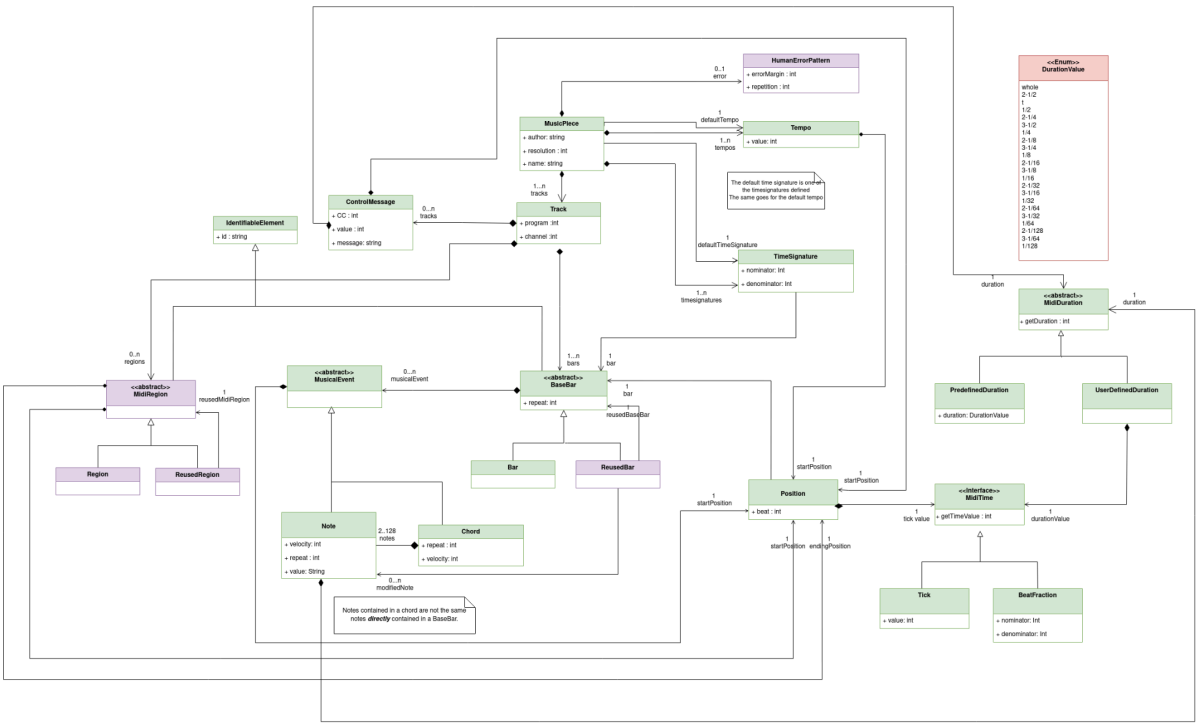


Figure : Métamodèle représentant le modèle de MusicML
(SVG)

Pour concevoir notre modèle de domaine, nous avons adopté une approche descendante “top-down”, débutant par le concept global englobant qui est “MusicPiece”, et progressant ensuite vers des détails plus spécifiques, jusqu'à atteindre les éléments les plus fins, ou "feuilles", de la structure conceptuelle. Il en résulte le métamodèle constitué des composantes vertes.

Après avoir établi cette structure, nous avons intégré des concepts supplémentaires à cette syntaxe abstraite intermédiaire pour supporter les features implémentés, ce sont les composants en violet.

Ajouts, modifications et concepts réifiés

MusicPiece, pistes et directives générales

Notre concept central est le **MusicPiece**, il réifie le concept de la pièce musicale en cours de composition. Un morceau de musique en général possède des descripteurs spécifiques tels que l'auteur, le nom et la résolution, en plus d'autres concepts propres à midi tels que les **directives générales** de la signature temporelle et du tempo, ainsi que les pistes qui seront jouées.

Nous nous sommes retrouvés à extraire ces concepts pour en faire des classes propres : une classe **Track**, une classe **Timesignature** et une classe **Tempo**.

Un morceau de musique est composé de plusieurs **pistes** *(Track)* de différents instruments joués simultanément sur un canal midi spécifique. Cela se traduit naturellement par une relation de composition entre les deux concepts MusicPiece et Track.

L'extraction de la **signature temporelle** *(Time Signature)* et du **tempo** *(Tempo)*, qui sont de simples valeurs, dans des classes propres a été faite non seulement pour mettre l'accent sur leur distinction en tant que concept de domaine, mais aussi pour soutenir la possibilité de changer le tempo ou la signature temporelle au fur et à mesure que le morceau de musique est joué.

En effet, ils sont liés au morceau de musique, car ils affectent toutes les pistes en même temps, et la définition de ce changement représente elle-même une *directive générale* à exécuter.

De la Piste à la Mesure : Supporter la position dans le temps logique

Le concept de position temporelle dans midi est représenté par le trio (bar, beat, tick), là où une piste est composée de plusieurs **mesures**. Ce concept de mesure a été directement réifié par l'ajout de la notion de **BaseBar**, en relation de composition avec la piste *(Track)* qui les contient. Le concept prend en charge la répétition séquentielle directe, grâce à son attribut "répétition".

Pour prendre en charge la possibilité de positionnement à l'intérieur d'une mesure, nous avons créé le concept de **position** *(Position)*, qui fait référence à la mesure *(BaseBar)* qu'il la concerne, précise le *beat* à l'intérieur de la mesure en question sous forme d'un entier et le décalage à l'intérieur de ce beat, représenté par des *ticks*.

Bien que le numéro du beat soit défini par un simple attribut dans la position, la valeur du tick est fournie par le concept avec lequel la position est en relation de composition : le **MidiTime**.

Nous avons choisi d'ajouter ce concept, au lieu d'une simple valeur entière de tick, pour offrir la possibilité de définir le décalage de façons différentes, en s'appuyant sur le ***polymorphisme*** du MidiTime : soit comme une valeur brute de tick *(Tick)*, soit comme une fraction du beat concerné *(BeatFraction)*. Par exemple : un décalage peut être égal à la moitié du beat.

Il est important de noter que le nombre de beat autorisés, et la valeur par défaut de la durée d'un beat à utiliser dans la fraction, sont extraits du numérateur et du dénominateur de la signature temporelle courante, respectivement.

Jusqu'à présent, la syntaxe abstraite devient très expressive et permet des comportements étranges tels que la spécification d'un beat qui dépasse le nombre de beat dans la signature temporelle, ou une quantité anormale de décalage de ticks qui n'est pas cohérente avec la résolution. C'est au validateurs de s'en occuper et de lancer une erreur en cas d'absurdité

Notes, accords et durées

En MIDI, on parle d'**événements musicaux** qui sont les **notes** à jouer. Pour réaliser cela, nous avons créé le concept d'**événement musical** *(MusicalEvent)*, contenu dans des mesures, d'où la relation de composition.

Ces événements musicaux possèdent une **position**, exploitant le concept que nous avons défini auparavant. Ils peuvent se présenter sous la forme d'une **note simple** *(Note)*, caractérisée par un pitch (sa hauteur tonale), une durée et une vélocité, ou sous la forme d'un **accord** (Chord)**,** qui est l'émission simultanée (ou quasi simultanée) de plusieurs notes, formant ainsi une composition de notes.

Concernant la durée, nous avons opté pour sa conceptualisation en tant qu'entité distincte, nommée **MidiDuration**. L'utilisation d'une classe séparée pour la durée offre une double approche : elle peut être exprimée en ***MidiTime*** *(UserDefinedDuration)*, avec des ticks comme unité de mesure, ou en utilisant des **durées musicales prédéfinies** *(PredefinedDuration)*. Ces durées sont inspirées des valeurs par défaut proposées par les claviers MIDI virtuels, basées sur des durées musicales standard telles que **la ronde, la blanche, la noire**, etc. L'idée est de permettre aux musiciens de spécifier la durée d'une note en termes musicaux conventionnels plutôt qu'en nombre de ticks.

Nous reconnaissons que le concept d'accord pourrait être techniquement réduit à une manipulation de la position des notes individuelles, et qu'il s'agit en quelque sorte d'une facilité syntaxique. Néanmoins, nous jugeons important de l'inclure dans notre modèle pour refléter fidèlement un concept essentiel du domaine musical.

De plus, notre modèle prend en charge des nuances dans le positionnement des notes au sein des accords. Les notes peuvent être jouées simultanément, partageant la position de l'accord, ou elles peuvent avoir des positions individuelles tout en étant regroupées dans un accord, permettant ainsi même d'avoir un décalage entre elles. Ce dernier cas conduit à la formation d'un **accord brisé,** également connu sous le nom **d'arpège** en musique. Dans les situations où les deux spécifications sont présentes, celle de la note l'emporte.

Réutilisation et modification des mesures :

Notre modèle sémantique intègre une fonctionnalité supplémentaire : **la capacité de réutiliser et de modifier des mesures musicales**. Cette fonctionnalité s'appuie sur une adaptation du modèle composite.

Le concept de **mesure** *(BaseBar)* que nous avons définie peut désormais se présenter sous deux formes : **une mesure standard** *(Bar)* ou **une mesure réutilisée** (*ReusedBar*). La mesure réutilisée fait référence à une mesure *(BaseBar)* existante. De plus, cette mesure de base est conçue comme un élément identifiable *(IdentifiableElement)*, doté d'un identifiant unique qui permet de la référencer.

Puisque le "BaseBar" est en relation de composition avec les événements musicaux, la "ReusedBar" ne réfère pas seulement une mesure existante ; elle peut également **incorporer** de nouveaux **événements musicaux**. Pour supporter également **la modification et la suppression des notes**, notre modèle permet également de référencer des notes déjà existantes et de les manipuler par dans la “ReusedBar”.

Cette approche ne se limite pas à la simple réutilisation de mesures ; elle permet également la réutilisation de mesures déjà utilisées, créant ainsi une chaîne de réutilisation. Cette flexibilité offre des possibilités étendues pour la manipulation et l'adaptation des structures musicales au sein de notre modèle.

Régions musicales dans les pistes :

La restriction de réutiliser uniquement une mesure à la fois nous a incités à introduire le concept de **région (***MidiRegion)*, offrant ainsi la possibilité de segmenter et réutiliser des morceaux spécifiques dans une piste avec n'importe quelle longueur.

Nous avons tiré inspiration du concept des régions MIDI du logiciel Logic Pro X, qui offre une représentation et une organisation des données MIDI au sein des pistes.

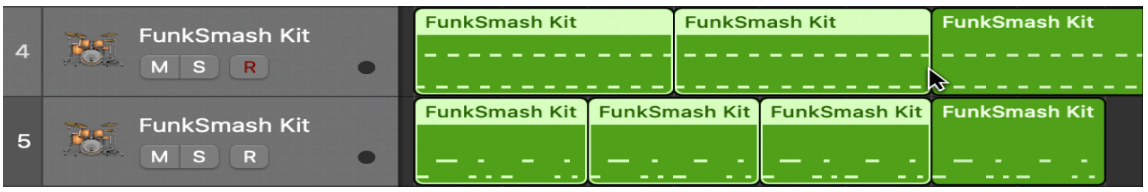


Figure 1 : Prise d’écran de réutilisation des régions dans Logic Pro X

Pour concrétiser le concept de régions dans notre modèle, nous avons introduit deux nouvelles entités : **Region** et **ReusedRegion**. Cette création suit la même logique de **référencement** que celle utilisée pour les barres, où un élément identifiable *(IdentifiableElement)* - dans ce cas, une région - est référencé.

Une **Region** est définie par **un point de début** et **un point de fin**, s'appuyant sur notre concept de **position**. Cette définition permet de spécifier clairement la portion de la track que la région couvre. La **ReusedRegion**, quant à elle, permet de réutiliser une région existante dans d'autres parties de la track.

Humanisation : erreur humaine

Pour soutenir l'humanisation d'une piste en intégrant **un motif récurrent et alterné d'erreurs,** telles que des **retards** et des **avances**, nous avons adopté le concept de *HumanError*. Ce concept est conçu pour concrétiser ces erreurs, définies par le délai et le

nombre de notes affectées à chaque occurrence. Cela réifie le concept d'avoir un motif où l'on est en avance sur le temps pour le nombre de notes précisé, puis en retard, et ainsi de suite, alternativement.

III. Syntaxe concrète sous forme de BNF

Pour la BNF, nous avons essayé de définir une BNF qui soit aussi intuitive et naturelle que possible pour un musicien, puisque nous utilisons un DSL externe. La voici :

```
<MusicPiece> ::= ("Live" <STRING>)?
                 "MusicPiece" <STRING>
                 ("Author" <STRING>)?
                 ("HumanError" <INT> "notes" <INT> "ms")?
                 ("TicksPerQuarterNote" <INT>)?
                 "DefaultTempo" <INT> "bpm"
                 ("Tempos" <Tempo>*)?
                 "DefaultTimeSignature" <INT> "/" <INT>
                 ("TimeSignatures" <TimeSignature>*)?
                 <SongDefinition>

<Tempo> ::= <INT> "bpm" <Position>

<TimeSignature> ::= <INT> "/" <INT> <INT>

<SongDefinition> ::= <Track>+ | <Sections>

<Track> ::= "Track" <STRING>
                 "Instrument" <STRING>
                 ("Channel" <INT>)?
                 <ControlMessage>*
                 <ContentDefinition>

<Sections> ::= "tracks :" <TrackDef>+
                        "sections :" <Section>+
                        "arrangement :" <STRING>+

<TrackDef> ::= "Track" <STRING>
                        "Instrument" <STRING>
                         ("Channel" <INT>)?

<Section> ::= "Section" <STRING> ":"
                         <SectionTrack>+

<SectionTrack> ::= "Track" <STRING>
                                <ControlMessage>*
```

```
                                    <ContentDefinition>

<ContentDefinition> ::= <PlayableBars> | <ArrangedBars>

<PlayableBars> ::= "|" <BaseBar>+

<ArrangedBars> ::= "bars :"
                                    "|" <BaseBar>+
                                    "regions :" <MidiRegion>+
                                    "arrangement :" <Arrangement>+

<Arrangement> ::= <ArrangedBar> | <ArrangedRegion>

<ArrangedBar> ::= "bar" <STRING> (",")?

<ArrangedRegion> ::= "region" <STRING> (<RegionStart>)? (",")?

<RegionStart> ::= <NegativeNotePosition> | <NotePosition>

<ControlMessage> ::= "CC :" "(" (<INT> | <STRING>) "," <INT> "," <Position> ")"

<BaseBar> ::= <Bar> | <ReusedBar> | <EmptyBar>

<Bar> ::= "Bar" <STRING> "->" <MusicalEvent>+ "|"

<ReusedBar> ::= "ReuseBar" <STRING> "use:" <STRING> (<INT> "times")?
                            ( "->" ("a:" "[" <MusicalEvent>+ "]")? ("r:" "[" <RemoveEvent>+ "]")?
("c:" "[" <ChangeEvent>+ "]")? )? "|"

<RemoveEvent> ::= "(" <STRING>* "," <NotePosition> ")" (",")?

<ChangeEvent> ::= "(" <STRING> "," <NotePosition> ")" "-->" <STRING> (",")?

<EmptyBar> ::= "EmptyBar" (<INT> "times")? "|"

<MusicalEvent> ::= <Note> | <Rest> | <Chord>

<Note> ::= "(" ("rep:" <INT> ",")? <STRING>+ ("," "p:" <NotePosition>)? "," ("d:")?
<MidiDuration> ("," "v:"  <INT>)? ")" (",")?

<Chord> ::= "Chord" "(" ("rep:" <INT> ",")? <Note>+ ("," "p:" <NotePosition>)? ("," "v:"
<INT>)? ")" (",")?

<MidiDuration> ::= <DurationValue> | "custom:" <MidiTime>

<MidiTime> ::= <Tick> | <BeatFraction>
```

```
<Tick> ::= "T" <INT>

<BeatFraction> ::= <INT> "/" <INT>

<DurationValue> ::= "whole" | "2-1/2" | "t" | "1/2" | "2-1/4" | "3-1/2" | "1/4" | "2-1/8" | "1/8" |
"2-1/16" | "3-1/8" | "1/16" | "2-1/32" | "3-1/16" | "1/32" | "2-1/64" | "3-1/32" | "1/64" |
"2-1/128" | "3-1/64" | "1/128" | "3-1/4"

<Position> ::= "(" <INT> "," <INT> "," <MidiTime> ")"

<NotePosition> ::= <INT> | "(" <INT> "," <MidiTime> ")"

<NegativeNotePosition> ::= "-" <NotePosition>

<Rest> ::= "(" "Rest" ("p:" <NotePosition> ",")? <MidiDuration> ")" (",")?

<MidiRegion> ::= "Region" <STRING> "->" "(" <Position> "," <Position> ")"

<STRING> ::= ( [a-z] | [A-Z] | [0-9] | "_" )+

<INT> ::= [0-9]+

<Comment> ::= "//" <STRING>
```

Figure 2 : BNF de MusicML en textX

On se trouve à ce niveau avec des concepts qui ne figurent pas explicitement dans notre modèle de domaine initial. Ces concepts ont été introduits principalement pour offrir du "sucre syntaxique". Nous détaillerons ces concepts supplémentaires dans la section suivante.

Description et critique du langage

Dans l'élaboration de notre langage spécifique au domaine (DSL), nous avons tiré parti de la flexibilité offerte par la création d'un DSL externe. Cette approche nous a permis de concevoir une syntaxe structurée en deux parties distinctes :

Une section "Header" : Cette partie contient des directives générales et une description globale. Elle est destinée à définir les paramètres et les informations qui s'appliquent à l'ensemble de la composition musicale.

On retrouve ici les mêmes concepts autour d'un MusicPiece dans le modèle du domaine : le **nom de la pièce musicale,** le **nom de l'auteur, la résolution temporelle (ticks per quarter note), tempo et signature temporelle par défauts et déclarations de changement de tempo et signature temporelle avec position de déclenchement.**

Ce qui est intéressant à expliciter est que pour la résolution de la pièce musicale nous avons prévu une valeur par défaut. Nous avons établi une résolution de 960 ticks par quart de note. Cette résolution détermine le niveau le plus fin de détail temporel dans le fichier. Bien que ce soit un détail subtil, il est important pour la précision du timing dans la musique. Nous avons compris ce besoin et nous avons opté pour cette valeur par défaut optimale afin de simplifier le processus pour l'utilisateur, qui n'a pas besoin de s'en préoccuper initialement.

Dans notre modèle, la section des métadonnées est conçue pour regrouper et clarifier tous les détails supplémentaires qui entourent la composition musicale, permettant ainsi aux utilisateurs de se concentrer sur la composition elle-même dans la section qui suit.

Pour les changements de tempo et de signature temporelle, nous avons adopté une méthode déclarative où l'utilisateur indique uniquement le point de déclenchement du changement. Cette approche est utile lorsque le changement est destiné à s'appliquer pour le reste de la piste, évitant ainsi à l'utilisateur la contrainte de devoir connaître et indiquer le moment exact de fin de la piste.

Une section de composition : Cette section est consacrée à détailler la composition musicale elle-même.

Dans la structuration de notre syntaxe, l'utilisateur a la possibilité de définir une chanson de deux manières : soit comme un ensemble de pistes individuelles, soit en utilisant des sections. Ceci permet d’avoir un mode de composition traditionnel, séparé de la fonctionnalité des sections que nous proposons.

Composition du morceau en pistes :

Examinons de plus près le scénario principal, celui de la définition de la chanson par le biais de pistes directement.

Pour leur définition, nous avons choisi également une approche déclarative qui facilite la configuration de leurs attributs : nom de la piste, instrument et canal.

Bien que le format MIDI identifie les instruments par des numéros de programme, nous permettons à l'utilisateur de spécifier l'instrument en langage naturel. Notre système est ensuite chargé de convertir cette désignation en le numéro de programme MIDI approprié.

De même pour réduire le travail du compositeur des valeurs par défaut s’applique en absence de spécification des canaux : Pour les instruments spécifiques comme la batterie, la valeur par défaut du canal est fixée à 10, tandis qu'elle est de 1 pour les autres instruments. Cette distinction reflète les conventions standards du MIDI et simplifie le processus de configuration pour l'utilisateur.

Il est également possible d'intégrer des messages de contrôle au sein des pistes. Ces messages peuvent être utilisés pour effectuer des ajustements tels que la modification de la hauteur (pitch) ou l'augmentation du volume. L'utilisateur fournit le nom du message de contrôle ; notre système se charge ensuite de le convertir en le numéro de code MIDI approprié.

Nous catégorisons toutes ces informations comme des métadonnées pour la piste concernée, c'est pour cela qu'on a essayé de réduire au maximum le nombre qu'il faut spécifier, et l'effort pour le faire aussi.

Or, il existe un attribut particulier dans la syntaxe de la piste qui n'est pas la même que celui dans le modèle de domaine. Il s'agit de la définition du contenu de la track : "ContentDefinition". Nous offrons deux possibilités pour procéder, soit en spécifiant un contenu qui est défini par PlayableBars, soit par ArrangedBars.

Playable Bars

En choisissant une définition de contenu "PlayableBars", nous choisissons de ne pas utiliser le concept de région. Dans ce cas la représentation visuelle de la piste dans notre modèle commence par une ligne horizontale "|", symbolisant le début de la piste. Elle est suivie par les mesures, qui sont également délimitées par des lignes horizontales "|". Cette structuration vise à maintenir une certaine familiarité avec l'apparence d'une partition musicale traditionnelle, facilitant ainsi la lecture et la compréhension par les utilisateurs habitués aux notations musicales classiques.

Dans notre modèle, chaque mesure possède un identifiant unique, facilitant sa réutilisation, et doit contenir au moins un événement musical.

Examinons de plus près le concept d'événement musical. Un événement musical peut être une note, un accord (comme défini dans notre modèle de domaine), ou un silence (Rest). Le silence est un ajout syntaxique pour indiquer une absence de son. Cette notion s'inspire de la définition MIDI, où une note jouée sans vélocité est considérée comme silencieuse.

Les notes peuvent être répétées, avec une valeur de répétition spécifiée, et peuvent avoir une position et une vélocité définies (par défaut, elles sont placées au premier beat et à une vélocité de 50%). Elles ont également une durée, exprimée en MidiDuration.

Dans notre syntaxe DSL, nous avons enrichi la manière dont les hauteurs de notes sont spécifiées, en les différenciant de la représentation numérique standard du MIDI. Au lieu d'utiliser des chiffres pour définir la hauteur des notes, les utilisateurs les fournissent sous forme de chaînes de caractères, et notre kernel se charge de valider ces chaînes. Ce choix rend la composition plus intuitive et accessible pour les musiciens, en leur permettant d'utiliser des notations familières plutôt que des valeurs numériques MIDI.

Cette approche permet de définir les pitchs en utilisant différentes notations musicales. Par exemple, ils peuvent utiliser la notation latine avec des indications d'octaves, comme "DO4", "RE3", "MI...", ou la notation américaine, comme "A3", "B4", "C4"... Cette flexibilité s'étend également à la notation de batterie.

Un accord peut aussi être répété, avoir une vélocité globale et une position spécifique, avec les mêmes valeurs par défaut, en plus des notes qu'il contient.

Pour une durée personnalisée, l'utilisateur indique "custom" et fournit un MidiTime, soit en fraction de beat, soit en ticks. Ceci est un peu trop verbeux, peut-être aurions-nous pu

trouver un meilleur moyen de différencier intuitivement une valeur personnalisée d'une valeur prédéfinie dans notre parsing.

Concernant la barre réutilisée, nous avons simplifié les instructions pour modifier les notes. Une barre réutilisée peut ne rien modifier, mais si des modifications sont nécessaires, un suffixe est utilisé pour l'indiquer : 'a' pour ajouter, 'c' pour changer, et 'r' pour enlever. Pour l'ajout, on se réfère à l'événement musical ; pour l'enlèvement, on vérifie l'existence de la note à la position donnée et on la supprime ; pour la mise à jour, on modifie uniquement le pitch du son. Actuellement, notre version de MusicML ne prend pas en charge la modification des accords, ce qui pourrait être une amélioration future.

Nous introduisons également différents éléments de sucre syntaxique, comme le concept d'EmptyBar. Ce concept, absent de notre modèle de domaine, représente une mesure sans note, équivalente à une durée de silence. Pour éviter la dissonance de définir une barre vide avec un identifiant et sans événement musical (qui pourrait entraîner une erreur de validation), nous avons inclus ce concept dans notre syntaxe. Cela aurait également été possible en donnant plus de flexibilité à la mesure elle-même.

Il est essentiel de distinguer que la position d'une note dans notre modèle diffère de la position globale. Dans notre syntaxe, nous introduisons le concept de "NotePosition", qui reconnaît que la position d'une note est relative à l'intérieur d'une mesure. Par conséquent, il n'est pas nécessaire de spécifier à nouveau cette position dans le contexte global de la composition en spécifiant le numéro du bar actuel.

Cette nuance est particulièrement importante car notre modèle de domaine, dans sa forme actuelle, est très expressif et permet de spécifier la position d'une note en incluant la mesure, en raison de son unique concept de "Position". Cependant, le "NotePosition" dans la syntaxe est conçu pour simplifier cette expression en se concentrant uniquement sur la position de la note au sein de sa mesure spécifique, sans nécessiter de références supplémentaires à la mesure globale. Cette approche vise à alléger la charge cognitive pour l'utilisateur lors de la spécification des positions des notes, en rendant le processus plus intuitif et moins encombré.

Arranged Bars

L'introduction de la définition de contenu sous forme d'"ArrangedBars" dans notre modèle offre à l'utilisateur une flexibilité dans la manière de structurer sa piste. Cette fonctionnalité permet à l'utilisateur de composer des mesures, de définir des régions spécifiques au sein de la track en utilisant des positions de début et de fin, et ensuite de faire référence à ces mesures et régions dans une section d'arrangement dédiée, située à la fin.

L'objectif principal de "ArrangedBars" est d'étendre les possibilités de structuration au niveau de la piste. Comme dans notre modèle de domaine, pour définir une région, l'utilisateur spécifie une position temporelle de début et une position temporelle de fin.

Toutefois, lorsqu'il procède à la création de l'arrangement de la piste, l'utilisateur ne se contente pas de référencer les mesures et les régions qu'il souhaite placer, mais il peut également positionner le point de départ à l'intérieur de ces dernières.

Nous sommes conscients qu'à ce stade, les possibilités de jouer avec la composition sont nombreuses et peut-être même déroutantes ou complexes à gérer, mais étant donné qu'il ne s'agit que d'options, elles n'ajoutent aucune contrainte ou surcharge pour le compositeur lui-même.

Composition du morceau en sections :

Revenant à notre option de définir une chanson par section, cette fonctionnalité n'est pas représentée dans le modèle métier car elle s'agit d'une option ajoutée à notre dsl pour permettre l'implémentation des pistes d'une manière différente.
Elle offre une approche alternative de la composition. Ici, le compositeur déclare d'abord les pistes qu'il prévoit d'utiliser dans sa composition, en utilisant le jeton "tracks :", puis il procède à la définition du contenu de ces pistes, organisé par sections.

Dans notre syntaxe DSL, nous avons maintenu une approche déclarative pour faciliter la compréhension et l'utilisation par l'utilisateur. Après avoir déclaré les pistes, l'utilisateur définit ensuite ses sections, comme le couplet, le refrain, ou le pont, et organise le contenu des pistes à l'intérieur de ces sections. Contrairement à la fonctionnalité des régions, il n'y a pas de positionnement spécifique à l'intérieur des sections.

L'avantage de cette séparation claire en sections est qu'elle permet au compositeur de se concentrer sur un aspect à la fois et de revenir facilement pour ajuster la composition. Cette structure facilite également la répétition des éléments dans la composition.

Cependant, cette simplicité a un coût. Nous reconnaissons que cette méthode n'est pas la norme et peut ne pas être immédiatement intuitive pour tous les utilisateurs. C'est pourquoi la fourniture d'exemples concrets et d'extensions de snippets pour ce DSL est cruciale. Ces ressources offrent une clarté supplémentaire et des points d'entrée pour explorer et expérimenter avec la composition. Cette nécessité découle de la nature "trop expressive" de notre DSL, qui, tout en offrant une grande flexibilité, peut également présenter des défis en termes d'accessibilité et de facilité d'utilisation pour certains utilisateurs.

IV. Description des scénarios et leurs mises en œuvre

Dans tous les scénarios, nous avons activé l'option "Live", qui lance une interface permettant d'écouter la musique et de jouer en même temps. Vous trouverez davantage d'explications sur cette option dans le paragraphe "Add-on MIDI Visualizer & Playern".

Scénarios basiques

Scénario - "Billie Jean" de Michael Jackson:

Tiré de la chanson "Billie Jean" de Michael Jackson, ce scénario requiert que le DSL développé permette la description de toutes les variations, incluant le remplissage de batterie introduisant le refrain.

Vous trouvez ici le scénario. Pour générer ce dernier, il vous suffit d'exécuter le script **scenario1.sh** à la racine du projet. Le fichier MIDI généré est localisé dans le dossier "**generated**" à la racine et est nommé BillieJean.mid.

Scénario - "Love Is All" de Roger Glover:

Ce scénario s'appuie sur la composition musicale "Love Is All". Il englobe la prise en charge de signatures temporelles distinctes et de variations de tempo à plusieurs emplacements au sein de la chanson.

Vous trouverez ici le scénario. Pour générer ce dernier, il vous suffit d'exécuter le script **scenario2.sh** à la racine du projet. Le fichier MIDI généré est localisé dans le dossier "**generated**" à la racine et est nommé LoveIsAll.mid.

Description des extensions

Support for bar modifications

Cette extension offre à l'utilisateur la possibilité de définir une mesure en se basant sur une mesure existante. Il explore diverses manipulations telles que l'ajout de nouvelles notes, les remplacements, les suppressions, etc.

À cet effet, nous avons rédigé deux scénarios : l'un montrant l'ajout de nouvelles notes et l'autre se concentrant sur les suppressions.

Dans le scénario de suppression de la note, nous réutilisons la première mesure en enlevant la note ‘DO2’.

```
Track Drum Instrument 'Acoustic Bass Drum'
```

```
| Bar id1 -> ('DO2' 'FA#2' , 1/8),  Chord(rep:2,( 'FA#2', p: (1, 1/2) , 1/8), ('FA#2' 'MI2', p: 2, 1/8), ('FA#2', p: (2, 1/2) , 1/8), ('DO2' 'FA#2', p: 3, 1/8) ) |
ReuseBar id2 use:id1 -> r:[('DO2', 1)] | ReuseBar id3 use:id1 31 times -> a:[(rep:9,'LA#4', 1/8)] | Bar id4 -> (rep:9, 'LA#4', 1/8), ('DO2' 'FA#2' , 1/8), Chord(( 'FA#2', p: (1, 1/2), 1/8),('FA#2' 'MI2', p: 2, 1/8), ('FA#2', p: (2, 1/2) , 1/8), ('DO2' 'FA#2', p: 3, 1/8) ),
Chord(( 'FA#2', p: (3, 1/2), 1/8), ('FA#2' 'MI2', p: 4, 1/8), ('LA#2' , p: (4, 1/2), 1/8), ('MI2' , p: (4, 3/4), 1/8) ) |  ReuseBar id5 use:id1  -> a:[(rep:9,'LA#4', 1/8)] |
```

Dans le scénario de remplacement de la note, nous réutilisons la première mesure en remplaçant la note ‘DO2’ par ‘FA4’

```
Track Drum Instrument 'Acoustic Bass Drum'  velocity 100

| Bar id1 -> ('DO2' 'FA#2' , 1/8),  Chord(rep:2,( 'FA#2', p: (1, 1/2) , 1/8), ('FA#2' 'MI2', p: 2, 1/8), ('FA#2', p: (2, 1/2) , 1/8), ('DO2' 'FA#2', p: 3, 1/8) )|
ReuseBar id2 use:id1 ->  c:[('DO2', 1) --> 'FA4']  | ReuseBar id3 use:id1 31 times -> a:[(rep:9,'LA#4', 1/8)] | Bar id4 -> (rep:9, 'LA#4', 1/8), ('DO2' 'FA#2' , 1/8), Chord(( 'FA#2', p: (1, 1/2), 1/8),('FA#2' 'MI2', p: 2, 1/8), ('FA#2', p: (2, 1/2) , 1/8), ('DO2' 'FA#2', p: 3, 1/8) ),
Chord(( 'FA#2', p: (3, 1/2), 1/8), ('FA#2' 'MI2', p: 4, 1/8), ('LA#2' , p: (4, 1/2), 1/8), ('MI2' , p: (4, 3/4), 1/8) ) |  ReuseBar id5 use:id1  -> a:[(rep:9,'LA#4', 1/8)] |
```

Vous trouverez ici le premier scénario et le deuxième scénario.

Pour générer les deux scénarios, exécutez simplement le script **scenario3.sh** à la racine du projet. Les deux fichiers MIDI générés se trouvent dans le dossier "generated" à la racine et sont nommés BillieJean_changeNote.mid et BillieJean_removeNote.mid.

Support for user interactive input

Dans ce scénario, l'utilisateur a la possibilité d'interagir de manière synchronisée avec la musique générée à l'aide du clavier. Une fois qu'un fichier .mid a été choisi via notre interface, la musique sera diffusée. Pour jouer simultanément, il suffit de choisir un instrument (numéro 1 sur la figure) et de cliquer sur le bouton "Modify Mapping" pour ajuster la correspondance entre les touches du clavier et les notes. La personnalisation de l'expérience musicale est encore étendue grâce à la possibilité de modifier les channels (3 sur la figure) et les drums (2 sur la figure). En appuyant sur le clavier, vous pourrez ainsi jouer de la musique.

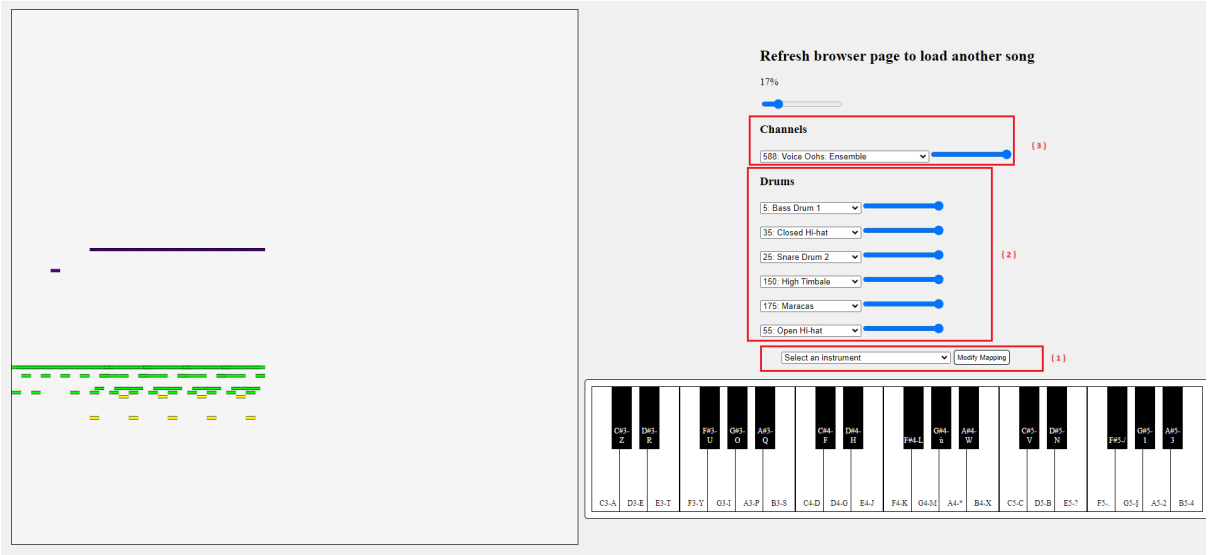


Figure 2 : Interface interactive visualisant le morceau composée

Support for human like errors

Cette extension permet à l'utilisateur de caractériser les erreurs de synchronisation spécifiques à certains batteurs, améliorant ainsi l'agrément musical. Intégrer cette extension offre à l'utilisateur la possibilité de définir l'erreur en précisant le nombre de notes ainsi que la marge, en millisecondes, à appliquer dans la piste entière pour ajouter ou retrancher cette durée à la position de la note :

HumanError 5 notes 200ms

Vous trouverez ici le scénario. Pour générer ce dernier, il vous suffit d'exécuter le script **scenario4.sh** à la racine du projet. Le fichier MIDI généré est localisé dans le dossier "**generated**" à la racine et est nommé LoveIsAll_humanError.mid.

Autres scénarios

Pitch bend

Dans ce scénario, nous avons implémenté le début de la piste de basse de “Billie Jean” avec un pitch bend ajouté grâce au commandes de contrôle (CC). À la mesure 1, sur le premier temps, le pitch bend est réglé à 100, puis il est ramené à 0 au début de la deuxième mesure.
Vous trouverez ici le scénario. Pour générer ce dernier, il vous suffit d'exécuter le script **scenario5.sh** à la racine du projet. Le fichier MIDI généré est localisé dans le dossier "**generated**" à la racine et est nommé BillieJean_pitchbend.mid.

Remarque: Nous avons constaté que la bibliothèque que nous avons utilisée pour créer notre interface graphique ne prend pas en charge le pitch bend. Pour confirmer son effet, nous avons joué le fichier musical généré sur une autre plateforme.

Utilisation des Régions

Dans ce scénario, nous avons implémenté le début de la piste de la batterie de “Billie Jean”. Les mesures et les régions ont été définies, puis réorganisées pour spécifier l'ordre dans lequel les éléments doivent être joués.

Vous trouverez ici le scénario. Pour générer ce dernier, il vous suffit d'exécuter le script **scenario6.sh** à la racine du projet. Le fichier MIDI généré est localisé dans le dossier "**generated**" à la racine et est nommé billieJean_regions.mid.

Utilisation des sections.

Dans ce scénario, nous avons implémenté le début de “Billie Jean “ dans deux sections distinctes, A et B, chacune avec des instructions spécifiques pour les pistes de batterie, basse et voix. L'arrangement final spécifie l'ordre dans lequel ces sections doivent être jouées.

Vous trouverez ici le scénario. Pour générer ce dernier, il vous suffit d'exécuter le script **scenario7.sh** à la racine du projet. Le fichier MIDI généré est localisé dans le dossier "**generated**" à la racine et est nommé billieJean_sections.mid.

V. Add-on : MIDI Visualizer & Player

Pour enrichir l'expérience utilisateur de notre langage MusicML, nous avons intégré un complément sous la forme d'un visualiseur MIDI. L'objectif est de permettre à l'utilisateur de voir sa création musicale prendre vie de manière interactive et immédiate.

Dans notre définition BNF (Backus-Naur Form), nous avons inclus une option "Live" qui peut être set à “on”. Lorsqu'un utilisateur active ce mode pendant la rédaction de sa partition en MusicML, la compilation du fichier .mml produit non seulement l'artefact MIDI correspondant, mais initie également le lancement d'un serveur. Ce serveur ouvre automatiquement un navigateur web, affichant une application web conçue pour lire directement le fichier MIDI généré.

Cette application web offre à l'utilisateur une visualisation dynamique des notes de sa composition. Elle permet également une interaction en temps réel avec l'œuvre : l'utilisateur peut ajuster le volume des différentes pistes, augmenter ou diminuer leur intensité, ou encore modifier le programme utilisé sur une piste spécifique. L'ajout de cette fonctionnalité vise à renforcer l'acceptation et l'adoption de MusicML en attirant l'utilisateur final vers un écosystème plus riche et interactif.

Figure : Interface pour téléverser le .midi générer

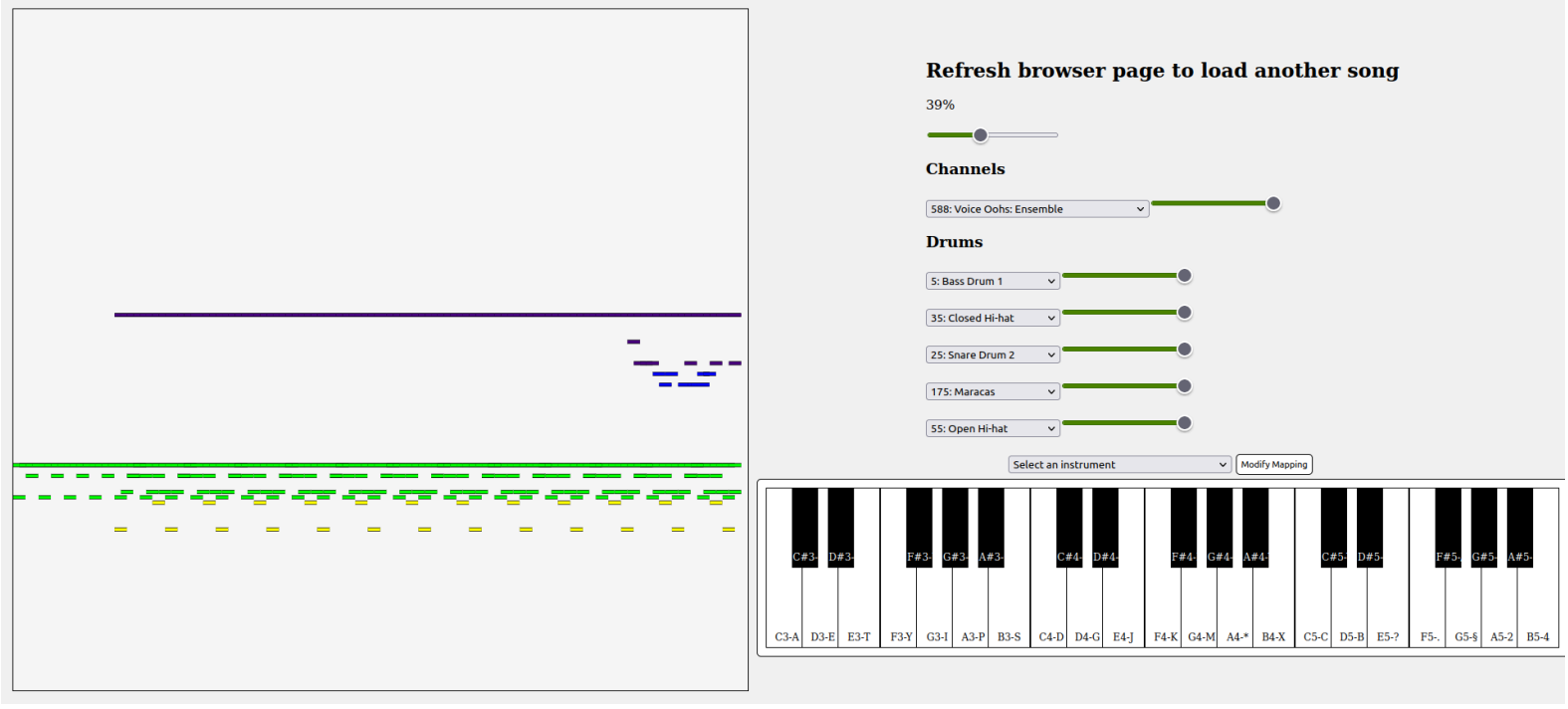


Figure 3 : Interface interactive visualisant le morceau composée

VI. Services Additionnels Fournies

Validateurs interne

TextX génère des erreurs en cas de détection d'erreurs de syntaxe ou de sémantique pendant l'analyse ou la construction du méta-modèle ou du modèle.

En cas d'erreur de syntaxe, une exception **TextXSyntaxError** est levée. Pour une erreur sémantique, c'est une exception **TextXSemanticError** qui est déclenchée. Les deux exceptions héritent de **TextXError**.

Ces exceptions comprennent un attribut message contenant le message d'erreur, ainsi que des attributs line et col indiquant respectivement la ligne et la colonne où l'erreur a été détectée.

Nous avons utilisé ces exceptions pour mettre en œuvre des validateurs dans notre kernel.

Exemple:

```
def compile_track(music_ml_model, music_ml_meta, track, midi_file, track_number,
ticks_to_add=0):
        track_settings(midi_file, music_ml_model, music_ml_meta, track, track_number)
        program_number = instrument_program_number(track.instrument)
        if program_number is None:
            raise TextXSemanticError('Instrument not found: ' + track.instrument,
**get_location(track))
```

La fonction **get_location()** permet de passer les informations nécessaires (nom de fichier, line, colonne) afin de localiser l'erreur dans notre fichier .mml.

```
Traceback (most recent call last):
  File "C:\Users\yimen\Documents\DSL-MusicML-TeamB\src\main.py", line 43, in <module>
    main()
  File "C:\Users\yimen\Documents\DSL-MusicML-TeamB\src\main.py", line 36, in main
    generate_midi_file(music_ml_meta, music_ml_model, '../generated/' + ml_file_name)
  File "C:\Users\yimen\Documents\DSL-MusicML-TeamB\src\generator.py", line 31, in generate_midi_file
    compile_track(music_ml_model, music_ml_meta, track, midi_file, i)
  File "C:\Users\yimen\Documents\DSL-MusicML-TeamB\src\compilers\track_compiler.py", line 10, in compile_track
    raise TextXSemanticError('Instrument not found: ' + track.instrument, **get_location(track))
textx.exceptions.TextXSemanticError: C:\Users\yimen\Documents\DSL-MusicML-TeamB\scenarios\BillieJean.mml:9:1: Instrument not found: bongo
```

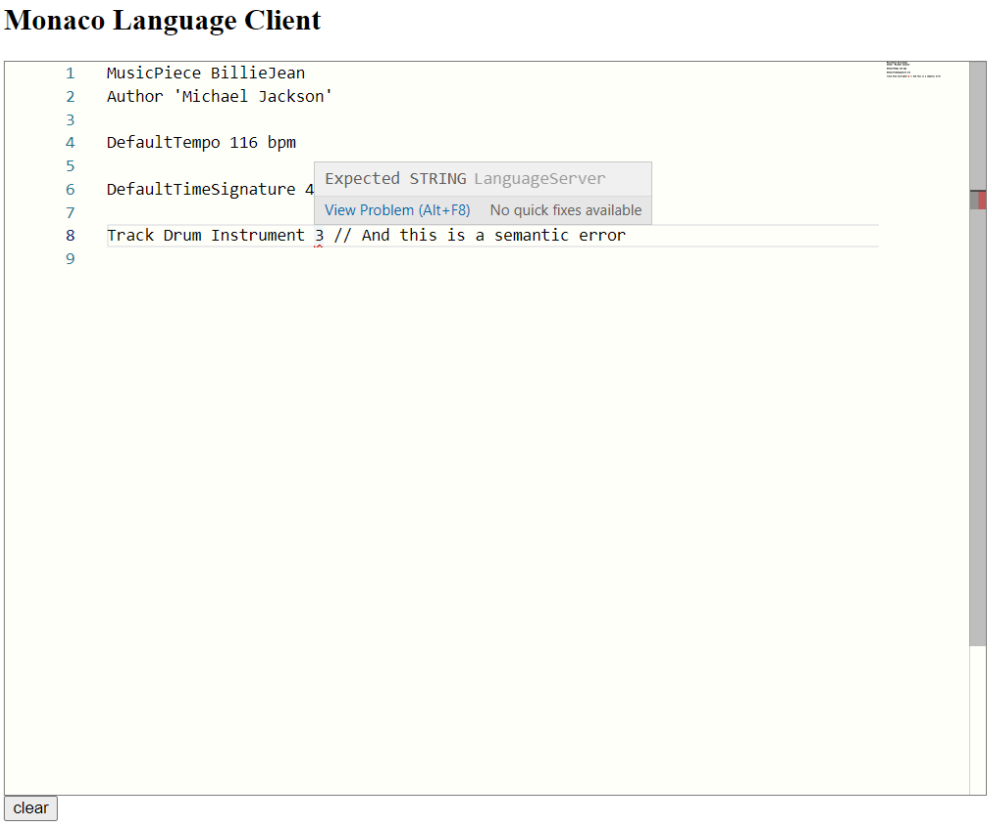
Figure 4 : Prise d'écran du message d'erreur lors de l'exécution d'un fichier mml.

```
1  Live on
2  MusicPiece BillieJean
3  Author 'Michael Jackson'
4
5  DefaultTempo 116 bpm
6
7  DefaultTimeSignature 4/4
8
9  Track Drum Instrument 'bongo'
```

Figure 5 : Prise d'écran d'un fichier mml avec un instrument non reconnu.

Validateur externe : l'éditeur Monaco

Nous avons amélioré l'expérience utilisateur de MusicML en intégrant l'éditeur Monaco. Celui-ci assure la vérification en temps réel de la validité du code saisi, signalant les erreurs et indiquant le type d'entrée attendu. Cette fonctionnalité vise à offrir une assistance proactive, simplifiant l'utilisation du langage.
Vous trouverez un README expliquant comment lancer l'éditeur Monaco.



Le bouton "Clear" est disponible pour supprimer le code saisi.

Nous avions d'abord envisagé d'offrir à l'utilisateur un éditeur classique (VSCode). Cependant, après avoir constaté que l'extension Textx n'était pas suffisamment maintenue, nous avons pris la décision de choisir Monaco Editor comme solution alternative. Cette décision a été motivée par le fait que Monaco Editor répond de manière optimale à nos besoins spécifiques.

VII. Analyse critique

Démarche

Pendant notre démarche, le principal obstacle que nous avons rencontré pour élaborer notre DSL était le manque de maîtrise musicale et de connaissance des règles musicales par tous les membres du groupe. Cela nous a pris du temps pour bien comprendre les principes fondamentaux de la musique et saisir les attentes du client.

Une fois cette étape franchie, nous avons entamé la phase de conception et de choix technologique.

Analyse du choix technologique

Notre choix technologique initial a été fortement influencé par l'écosystème disponible. Nous avons opté pour Langium en raison de notre familiarité avec ses avantages, ses licences et ses capacités, qui s'alignent parfaitement avec notre objectif de créer un DSL externe expressif, complété par une extension pour VSCode afin de faciliter son utilisation. De plus, nous avons découvert la bibliothèque midi-writer-js, qui nous a permis de jouer les sons et de créer un scénario de base.

Cependant, après avoir acquis une meilleure compréhension de la musique et avoir étendu notre projet avec les différents scénarios et extensions, nous avons dû nous tourner vers Python en raison du manque de bibliothèques adéquates en TypeScript , notamment midi-writer-js. Python bénéficie d'une communauté riche et active, offrant ainsi un large éventail de bibliothèques pour mettre en œuvre notre noyau compatible avec le MIDI de MusicML. Nous avons finalement utilisé la bibliothèque Python midiutil, qui répond parfaitement aux différentes fonctionnalités et extensions du sujet.

Pour l'implémentation du DSL lui-même, nous avons choisi la bibliothèque TextX, compatible avec Python. TextX, sous licence MIT sans restriction, est relativement bien maintenue. Toutefois, elle souffre d'un manque de popularité et d'un nombre restreint de contributeurs, ce qui pose un risque en termes de viabilité et de support à long terme. Une alternative aurait été de maintenir Langium pour le Language Server Protocol (LSP) tout en utilisant le noyau Python en parallèle, mais des contraintes de temps nous ont empêchés de revenir sur cette décision.

En rétrospective, bien que nous ayons manqué certaines opportunités offertes par Langium tel que le validateur et l’extension VSCode, l'utilisation de TextX s'est avérée être une solution satisfaisante, d'autant plus que nous avons trouvé un éditeur compatible Monaco pour l'accompagner.

Forces, faiblesses et points d’amélioration

Nous estimons que notre DSL (Domain-Specific Language) est suffisamment clair et pertinent. Nous prenons en charge les détails minutieux, tels que la position basée sur le

temps musical (beat) et un offset qui peut être exprimé par une fraction de beat ou un nombre de ticks, afin d'assurer une gestion précise et complète.
La clarté de la syntaxe de notre langage offre aux utilisateurs la possibilité de se concentrer pleinement sur les aspects musicaux spécifiques, éliminant ainsi toute confusion causée par une syntaxe complexe ou des fonctionnalités superflues.

L'utilisation de régions pour composer une piste permet d'identifier les parties musicales pouvant être répétées et de les organiser avec d'autres mesures. Nous avons introduit un attribut "start" pour organiser les régions, offrant ainsi la possibilité de la positionner avec un décalage à droite ou à gauche par rapport aux mesures. Cependant, lorsqu'on souhaite positionner le contenu d'une mesure spécifique après une région, il est inévitable que cela se fasse au début de la mesure suivante, créant éventuellement un espace non désiré entre le début de la mesure et la fin de la région.

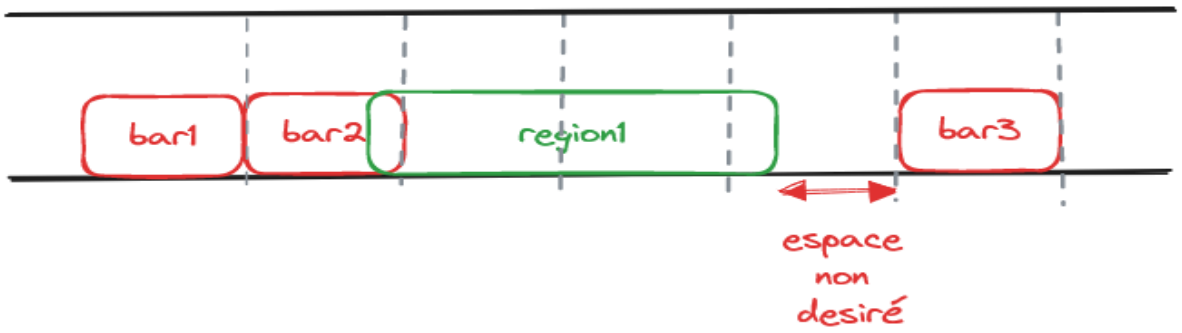


Figure 6: représentation d'une piste avec une région placée avec un décalage à droite

Une solution envisageable à ce problème aurait été d'ajouter un attribut de décalage pour le positionnement des mesures, ou de veiller à ce que les notes de la barre soient automatiquement positionnées immédiatement après la région, même si une se termine au milieu de la mesure précédente.

Enfin, des améliorations et des précisions ont été ajoutées, en particulier pendant la période où nous avions déjà une maîtrise préalable du domaine (période des vacances). En raison des contraintes temporelles liées à la période d'examen, nous estimons que nous aurions pu intégrer davantage d'améliorations au projet. Par exemple, l'extension de gestion des erreurs humaines pourrait être améliorée en injectant l'erreur dans une note spécifique, ce qui pourrait être intéressant pour l'utilisateur afin d'obtenir plus de personnalisation.

VIII. Responsabilité des membres de l'équipe

Nadim et Sourour ont collaboré sur l'implémentation des scénarios de base et l'intégration de l'éditeur Monaco, tandis que Badr et Imene ont pris en charge le développement des extensions. Nous avons tous participé activement à l'amélioration globale du projet.