Rapport Projet DSL MusicML

Auteurs :
Jules Prince - Théophile Yvars - Hamza Zoubair - Roméo David Amedomey

20 janvier 2024

Table des matières

1 Introduction

Dans le contexte du cours sur les DSL (Domain Specific Language), nous avons été assignés à la création d'un DSL dédié à la composition musicale. L'objectif de ce langage spécifique au domaine est de permettre aux compositeurs d'exprimer leurs idées musicales de manière structurée. Notre choix s'est porté sur une représentation en format MIDI pour notre projet.

Ce rapport a pour but de décrire de manière objective le langage que nous avons élaboré, ainsi que les fonctionnalités qu'il propose. Nous présenterons les différentes composantes du DSL, exposant les caractéristiques techniques et les choix de conception qui ont été faits au cours du développement. L'objectif ici est de fournir une analyse du DSL créé, mettant en lumière ses points forts tous comme ses limites.

2 Description du langage

2.1 Domaine de classe sous forme de diagramme

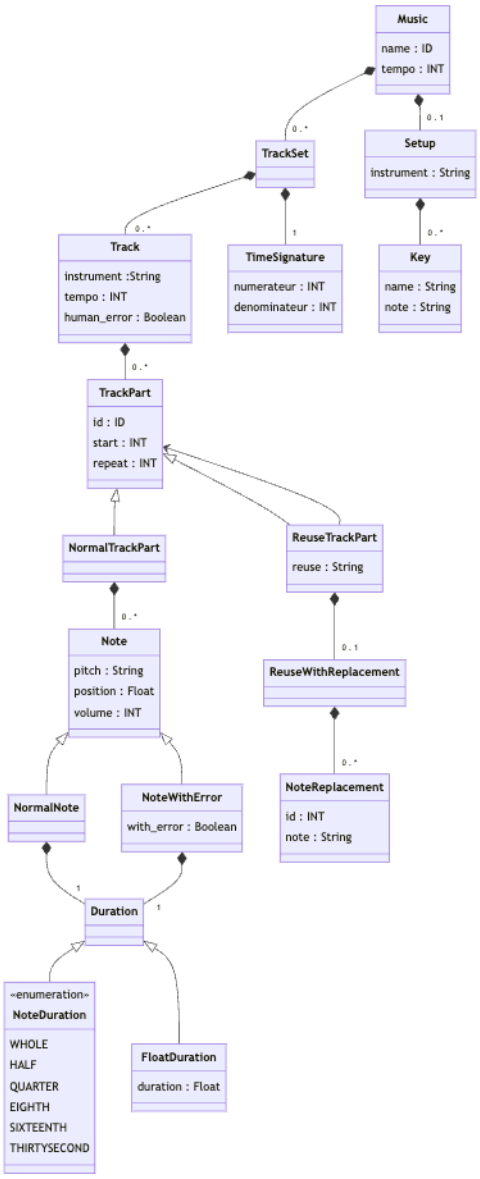


FIGURE 2.1.1 – Diagramme de classe

Vous trouverez ce diagramme ici .

L'objet racine dans notre DSL est représenté par la classe Music. Nous pouvons spécifier le tempo du son à l'aide de l'attribut "tempo". Chaque musique est composée d'un TrackSet, ce dernier représente une liste de pistes (Track) avec une signature rythmique. La création d'une partie sonore avec une nouvelle signature rythmique nécessite la création d'un nouveau TrackSet. Ce choix nous permet d'isoler les pistes sous la même signature rythmique.

Le TrackSet lui-même est composé d'une liste de Track. Ce dernier est défini par un instrument et un tempo. La répétition du tempo dans l'objet Track a pour raison de pouvoir le changer sous la même signature

rythmique. Nous pouvons introduire l'erreur humaine dans le Track en affectant à l'attribut "human error" la valeur true.

Dans un Track, on peut définir des TrackPart, chacune représentant une partie de la piste (exemple : mesure, couplet, refrain...). L'attribut "start" permet de spécifier le numéro de la mesure dans laquelle le TrackPart commence. Ce choix donne la liberté aux utilisateurs de choisir le commencement d'une TrackPart sans définir des silences. Par exemple, définir des notes pour la mesure 1 et ensuite définir la mesure 20 dans un autre TrackPart. L'attribut "repeat" permet de glisser le TrackPart sur les prochaines mesures. Nous avons mis en place deux types de TrackPart :

Le premier est NormalTrackPart qui est composé de notes. Chaque note est définie par le pitch, la durée de la note qui peut être une chaîne de caractères pour les durées classiques (whole, half, quarter, eighth, sixteenth, thirtysecond), comme elle peut être un float pour définir une durée quelconque, enfin, on définit le volume qui est la vélocité de la note. Nous disposons également d'un type de notes qui est NoteWithError, il permet d'injecter une erreur sur une note précise (l'erreur portera sur le temps de début et la vélocité).

Nous avons mis en place un mécanisme de réutilisation de TrackPart à travers le type ReuseTrackPart. On peut cibler un TrackPart à réutiliser à l'aide de l'attribut "reuse" en mettant son id. De plus, nous pouvons réutiliser un TrackPart en remplaçant des notes. L'objet NoteRemplacement permet de répondre à ce besoin en spécifiant le numéro de la note dans le TrackPart d'origine et la note avec laquelle on veut la remplacer.

Pour supporter la fonctionnalité de musique interactive depuis le clavier, nous avons ajouté la classe Setup. Cette classe permet de définir l'instrument que l'on veut utiliser et de faire le mapping entre les touches du clavier et les notes. La fonctionnalité de musique interactive permet à l'utilisateur d'écrire la musique jouée sur le clavier directement dans un fichier de scénario. Il est donc possible de composer de la musique directement avec le clavier et de la rejouer plus tard.

2.2 La syntaxe représenté sous forme de BNF

La grammaire de notre langage est accessible ici :

```
grammar Musicae

entry Music:
    'music' name=ID '{'
        ('tempo' tempo=INT)?
        trackSet+=TrackSet*
        (setup=Setup)?
    '}';

TrackSet:
    time_signature=TimeSignature
    track+=Track*;

Setup:
    'setup' '{'
        'instrument' instrument=Instrument
        'keys' '{'
            keys+=Key*
        '}'
    '}';

Track:
    'track' id=INT '{'
        'instrument' instrument=Instrument
        ('human_error' human_error=BOOLEAN)?
        parts+=TrackPart*
    '}';

TrackPart:
    'trackPart' id=ID '{'
        'start' start=INT
        ('repeat' repeat=INT)?
        notes+=Note*
        ('reuse' reuse=STRING)?
    '}';

Key:
    name=STRING '=>' note=STRING (',')?;


Note :
    NormalNote | NoteWithError;

NormalNote:
    pitch=STRING ',' position=Float ',' duration=Float ',' volume=INT ;

NoteWithError:
    pitch=STRING ',' position=Float ',' duration=Float ',' volume=INT  'human_error' with_error=BOOLEAN ;

```

```
49
50  Float:
51      n1=INT '.' n2=INT;
52
53  TimeSignature :
54      'time_signature' '{'
55          numerator=INT "," denominator=INT
56      '}';
57
58  Instrument:
59      name =STRING;
60
61
62  terminal BOOLEAN returns boolean: 'true' | 'false';
63  hidden terminal WS: /\s+/;
64  terminal ID: /[_a-zA-Z][\w_]*/;
65  terminal INT returns number: /[0-9]+/;
66  terminal FLOAT returns number: /[0-9]+\.[0-9]+/;
67
68  terminal STRING: /"(\\.|[^"\\])*"|'(\\.|[^'\\])*'/;
69
70
71  hidden terminal ML_COMMENT: /\/\*[\s\S]*?\*\//;
72  hidden terminal SL_COMMENT: /\/\/[^\n\r]*/;
73
```

FIGURE 2.2.1 – Enter Caption

2.2.1 Explication de la grammaire :

label=•**Music** : Représente une déclaration sonore avec un nom (ID), éventuellement un tempo, un ensemble de pistes (TrackSet), et facultativement une configuration du clavier spécifiant des notes spécifiques (Setup). **TrackSet** : Il s'agit d'un ensemble de pistes partageant la même signature rythmique. **Track** : Représente une piste musicale associée à un instrument. Il est possible de redéfinir le tempo en cas de changement de rythme sous la même signature. Si le tempo n'est pas spécifié, il est considéré équivalent à celui déclaré dans l'objet racine Music. L'attribut "human error" permet d'introduire une erreur humaine. On peut également définir des TrackPart. **TrackPart** : Soit la définition d'une nouvelle mesure ou partie de la piste, soit la réutilisation d'un TrackPart existant. L'attribut "start" (numéro de mesure de début) est défini, et l'attribut "repeat" permet de répéter le TrackPart sur un certain nombre de mesures selon la signature rythmique. Si "repeat" n'est pas défini, la répétition est ignorée. **NormalTrackPart** : Partie de la piste comprenant des notes. **ReuseTrackPart** : TrackPart qui réutilise un autre TrackPart en spécifiant son ID avec l'attribut "reuse". Il peut contenir un objet "ReuseWithReplacement" pour réutiliser un TrackPart en changeant des notes. **NormalNote** : Représente une note avec un nom, une durée, une durée de note et un volume. L'énumération "NoteDuration" peut être utilisée pour définir des durées classiques, sinon l'utilisateur peut saisir un nombre décimal comme durée. **NoteWithError** : Représente une note avec une erreur humaine. **Setup** : Définit un instrument et la correspondance entre les touches du clavier et les notes. Il contient plusieurs correspondances entre les touches et les notes. **Key** : Définit la correspondance entre une touche et une note.

2.2.2 Analyse critique de la grammaire :

Points positifs :

Notre grammaire se distingue par sa simplicité pour définir une composition musicale. Pour créer une piste avec un nouvel instrument, il suffit de définir un Track et de spécifier un instrument. Le musicien peut ensuite structurer son Track en définissant différentes parties (exemple : refrain, couplets, etc.) grâce à la création de nouveaux TrackPart. L'utilisation d'identifiants uniques pour les TrackPart permet leur réutilisation dans d'autres contextes.

Concernant l'introduction des notes, nous avons cherché à maximiser l'expressivité tout en simplifiant l'écriture au maximum. Il suffit d'une seule ligne pour décrire les caractéristiques d'une note (exemple : "F2", 0.0, 1.0, 90). Cette simplicité d'écriture revêt une importance particulière pour les musiciens, étant donné que chaque TrackPart peut contenir un grand nombre de notes. La possibilité de définir des durées avec des valeurs décimales offre une flexibilité, permettant de ne pas se limiter aux durées classiques.

L'ajout de l'attribut "start" dans la définition des TrackPart simplifie le processus, le musicien n'ayant pas besoin de spécifier des silences. Par exemple, s'il souhaite ajouter des notes dans la mesure 1 et ensuite créer des notes dans la mesure 20, il peut le faire simplement en attribuant au deuxième TrackPart la valeur 19 pour l'attribut "start".

Points négatifs :

Parmi les aspects défavorables de notre grammaire, on trouve l'obligation de spécifier la mesure de début pour chaque TrackPart. Si un musicien crée de nombreuses TrackParts, cela peut poser des difficultés pour mémoriser la mesure en cours. Cependant, cette approche offre la possibilité de sauter plusieurs mesures sans avoir à déclarer explicitement les silences.

La structure entre NormalTrackPart et ReuseTrackPart est similaire. Il aurait été envisageable de regrouper les éléments communs dans une seule règle et d'utiliser des options pour distinguer les cas spécifiques. Il existe des points où la factorisation du code aurait été possible.

Pour utiliser ce DSL, l'installation de bibliothèques en JavaScript et Python est nécessaire. On pourrait envisager de containeriser cette application avec Docker, assurant ainsi que toutes les dépendances soient incluses dans le package. Cependant, cela suppose que l'utilisateur ait déjà installé Docker. Une alternative serait d'héberger l'application dans le cloud et de fournir à l'utilisateur une interface pour créer et exécuter ses scénarios.

Le volume et la durée se répètent fréquemment. Il aurait été possible de permettre à l'utilisateur de ne pas les renseigner, récupérant ainsi la dernière valeur configurée. Cela aurait permis d'économiser du temps pour l'utilisateur.

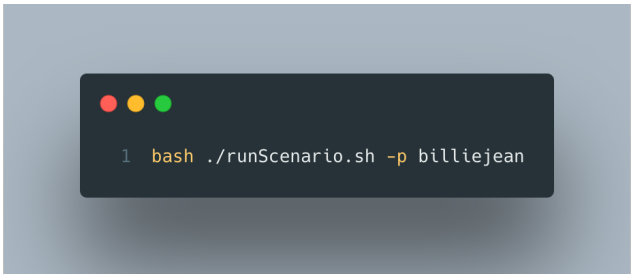
3 Extensions et Add-ons :

3.1 Add-ons

3.1.1 Jouer après génération

Pour améliorer le confort d'utilisation de Musicae, nous avons ajouté une fonctionnalité pratique qui permet de jouer automatiquement le fichier MIDI dès sa génération. Cela offre une expérience plus immersive et permet aux utilisateurs de vérifier rapidement le résultat musical sans avoir à lancer manuellement le fichier.

Pour activer cette fonctionnalité, il vous suffit d'ajouter le paramètre "-p" lors de la génération du fichier MIDI. Ainsi, votre commande de génération ressemblera à quelque chose comme ceci :



3.1.2 Extension vscode

Notre langage propose une extension vscode pour fournir de l'assistance à l'utilisateur lorsqu'il écrit de la musique, ce qui rend notre langage plus facile d'utilisation. Nous avons également ajouté un bouton qui aurait permis d'exécuter le scénario en un clic, mais cette fonctionnalité n'a malheureusement pas été complétée à ce jour.

3.2 Extensions

3.2.1 Interactive input

C'est l'une des extensions que nous avons choisi de développer. Cette extension permet à l'utilisateur de configurer son clavier comme celui d'un piano, lui permettant de jouer une note à chaque clic. Il est important de noter que la durée de la note dépend de la durée du clic, reproduisant ainsi le comportement d'un vrai piano. Cette fonctionnalité est configurable pour n'importe quel instrument, le piano n'étant qu'un exemple.

Une fois que l'utilisateur a terminé de jouer de la musique interactive, celle-ci est enregistrée dans un fichier de scénario spécifique à notre langage. Cela lui offre la possibilité d'exécuter le scénario pour entendre le son ou apporter des modifications ultérieures. Le choix de sauvegarder la musique directement dans un fichier de scénario a été fait délibérément, car nous souhaitons offrir à l'utilisateur la facilité de modifier sa composition de manière fluide avec notre langage. Cela aurait été moins pratique si le son avait été écrit directement, par exemple.

3.2.2 Bar modification

Nous avons décidé de mettre en place cette fonctionnalité pour enrichir notre DSL. La consigne initiale était de définir une mesure à partir de la précédente, avec la possibilité de modifier des notes. Comme notre

grammaire supporte déjà la réutilisation des TrackPart, nous l'avons adaptée pour permettre la réutilisation de n'importe quel TrackPart, pas uniquement du précédent. Le musicien n'a qu'à mentionner l'ID du TrackPart qu'il souhaite réutiliser. Pour les remplacements de notes, il suffit de récupérer la position de la note dans le TrackPart d'origine, puis d'indiquer la note de remplacement. Comme expliqué précédemment, cela permet de différencier les notes similaires. La position de la note dans ce contexte correspond à l'ordre de déclaration dans le TrackPart.

Exemple d'utilisation :

```
trackPart part2{
    start 5
    reuse "part1"
    replace {
        note 0 => "bd"
        note 1 => "cc"
        note 3 => "oh"
    }
```

FIGURE 3.2.1 – Enter Caption

Le scénario exploitant cette extension est explicité dans le paragraphe 4.4.

3.2.3 Erreur humaine

Nous avons implémenté cette fonctionnalité avec deux possibilités :

1. ~~1.~~ Injection de l'erreur dans une piste entière.
2. Injection de l'erreur dans une note précise.

Nous avons jugé intéressant d'injecter l'erreur dans des notes spécifiques, permettant ainsi au musicien de personnaliser l'erreur humaine sur des notes plutôt que sur l'ensemble de la piste.

Cela se réalise simplement en attribuant la valeur true à l'attribut **human error** dans le Track et dans la note. Nous avons configuré notre DSL pour générer une incertitude qui sera injectée dans les notes. Nous avons considéré que l'erreur humaine est significative pour le temps de début et la vélocité de la note. Par conséquent, l'erreur va décaler ces deux valeurs avec une incertitude aléatoire.

Le scénario exploitant cette extension est explicité dans le paragraphe suivant, 4.3.

4 Exemples de scenarios implémenté par notre DSL

4.1 Billie jean

```
music billiejean {
    tempo 118

    time_signature {
        4, 4
    }

    track {
        instrument "DRUM"
        trackPart track_1{
            start 0
            repeat 8
            "ch", 0.0, 1.0, 90
            "bd", 0.0, 1.0, 90
            "ch", 0.5, 1.0, 90
            "sd", 1.0 , 1.0 , 90
            "ch", 1.0, 1.0, 90
            "ch", 1.5, 1.0, 90
            "bd", 2.0, 1.0, 95
            "ch", 2.0, 1.0, 95
            "ch", 2.5, 1.0, 95
            "sd", 3.0 , 1.0 , 90
            "ch", 3.0, 1.0, 95
            "ch", 3.5, 1.0, 95
        }
    }

    track  {
            instrument "BASS"
            trackPart track_1{
                start 3
                repeat 4
                "F#2", 0.0, 1.0, 90
                "D2", 0.5, 1.0, 90
                "E2", 1.0, 1.0, 90
                "C#2", 1.5 , 1.0 , 90
                "D2", 2.0, 1.0, 90
                "F#2", 2.5, 1.0, 90
                "A2", 3.0, 1.0, 95
                "E2", 3.5, 1.0, 95
            }
        }
}
```

Url du scénario

Fichier Midi

4.2 Love is all

```
music billiejean {
    tempo 118

    time_signature {
        4, 4
    }

    track {
        instrument "DRUM"
        trackPart track_1{
            start 0
            repeat 8
            "ch", 0.0, 1.0, 90
            "bd", 0.0, 1.0, 90
            "ch", 0.5, 1.0, 90
            "sd", 1.0 , 1.0 , 90
            "ch", 1.0, 1.0, 90
            "ch", 1.5, 1.0, 90
            "bd", 2.0, 1.0, 95
            "ch", 2.0, 1.0, 95
            "ch", 2.5, 1.0, 95
            "sd", 3.0 , 1.0 , 90
            "ch", 3.0, 1.0, 95
            "ch", 3.5, 1.0, 95
        }
    }

    track  {
            instrument "BASS"
            trackPart track_1{
                start 3
                repeat 4
                "F#2", 0.0, 1.0, 90
                "D2", 0.5, 1.0, 90
                "E2", 1.0, 1.0, 90
                "C#2", 1.5 , 1.0 , 90
                "D2", 2.0, 1.0, 90
                "F#2", 2.5, 1.0, 90
                "A2", 3.0, 1.0, 95
                "E2", 3.5, 1.0, 95
            }
        }
}
```

Url du scénario

Fichier Midi

4.3 Billie jean joué par des "humains"

```
music billiejean_with_error {
    tempo 118
    time_signature {
        4, 4
    }
    track{
        instrument "DRUM"
        human_error true
        trackPart track_1{
            start 0
            repeat 8
            "ch", 0.0, 1.0, 90
            "bd", 0.0, 1.0, 90
            "ch", 0.5, 1.0, 90
            "sd", 1.0,1.0 , 90
            "ch", 1.0, 1.0, 90
            "ch", 1.5, 1.0, 90
            "bd", 2.0, 1.0, 95
            "ch", 2.0, 1.0, 95
            "ch", 2.5, 1.0, 95
            "sd", 3.0, 1.0, 90
            "ch", 3.0, 1.0, 95
            "ch", 3.5, 1.0, 95
        }
    }
}
```

Url du scénario Fichier Midi

Dans ce scénario, nous interprétons les huit premières mesures de la chanson "Billie Jean" en introduisant une erreur humaine. L'attribut **human error** est utilisé pour indiquer que toutes les notes doivent être jouées avec une erreur. Cette erreur affectera le temps de début de la note et son volume.

4.4 Exploitation du mécanisme de bar modification

```
music song_with_bar_modification {
    tempo 110
    time_signature {
        4, 4
    }
    track {
        instrument "DRUM"
        trackPart part1{
            start 0
            repeat 4
            "ch", 0.0, quarter, 90
            "bd", 0.0, 1.0, 90
            "ch", 0.5, 1.0, 90
            "sd", 1.0,1.0 , 90
        }

        trackPart part2{
            start 5
```

```
            reuse "part1"
            replace {
                note 0 => "bd"
                note 1 => "cc"
                note 3 => "oh"
            }


        }
    }
```

Url du scénario

Fichier Midi

Ce scénario vise à démontrer la fonctionnalité **Support for Bar Modification**. Pour ce son, nous avons extrait quelques notes du premier temps de la musique "Billie Jean". Par la suite, nous avons réutilisé cette mesure dans le deuxième TrackPart. Le deuxième TrackPart commence à la 6ème mesure (où 0 signifie la première mesure), et nous réutilisons la première mesure en modifiant 3 notes. Chaque indice fait référence à l'ordre de la note dans le TrackPart d'origine. La présence des indices (0, 1, 3) nous permet d'éviter les conflits en cas de notes répétées (par exemple, snare drum à deux temps différents). De plus, l'utilisateur n'a qu'à réécrire les notes qu'il souhaite remplacer.

4.5 Clavier numérique

```
music interactivePianoSetup {
    setup {
       instrument 'PIANO'
       keys {
        "A" => "C4"  // Do
        "Z" => "D4"  // Re
        "E" => "E4"  // Mi
        "R" => "F4"  // Fa
        "T" => "G4"  // Sol
        "Y" => "A4"  // La
        "U" => "B4" // Si
        "I" => "C5"
       }
    }
}
```

Ce scénario permet à l'utilisateur de jour du piano avec les touches AZERTYUI de son clavier. A la fin du scénario, l'utilisateur appui sur la touche ESC pour arrêter. Un fichier de scénario se crée automatiquement. Ce dernier contient la e sous forme de scénario que l'utilisateur vient de jouer. Il pourra aisément le modifier ou l'exécuter ce scénario pour obtenir le fichier midi.

Url du scénario

Fichier Midi

4.6 Son avec changement de signature

```
music time_signature_change {
    tempo 120
    time_signature {
        4, 4
    }
    track {
        instrument "PIANO"
        trackPart part1{
            start 0
            repeat 1
            "E4", 0.0, quarter, 90
            "D4", 1.0, quarter, 90
            "C4", 2.0, quarter, 90
            "D4", 3.0 , quarter , 90
        }
    }

    time_signature {
        3, 4
    }
    track {
        instrument "DRUM"
        trackPart part2{
            start 1
            repeat 1
            "ch", 0.0, quarter, 90
            "bd", 1.0, 1.0, 90
        }
    }
}
```

Ce scénario permet de démontrer le changement de signature entre les différentes pistes. La première piste est jouée avec une signature 4/4. Ensuite, nous définissons un nouveau TrackSet avec la signature 3/4, suivi des pistes sous cette nouvelle signature. Il est même possible de changer d'instrument entre les signatures temporelles. L'expressivité est très simple puisqu'il suffit de définir la nouvelle signature et de poursuivre la définition des pistes.

Url du scénario

Fichier Midi

5 Analyse critique de notre DSL

5.1 Implementation du DSL

Il est difficile de proposer un DSL pour répondre au besoin d'un musicien. Etre en compétition avec un éditeur graphique est complexe. Il est plus rapide de réaliser une musique avec l'un des logiciels que d'écrire les notes. Notre problématique est de proposer un DSL qui soit le moin contraignant possible tout en offrant le plus possible de configuration. Le monde de la musique peut être complexe, et est sujet à tout un tas de varibale pouvant grandement influencer le son.

Pour définir une musique, l'artiste la décompose en trackSet. Chaque trackSet définit un ensemble d'instruments et une time signature. Nous sommes partie sur l'hypothese forte qu'un time signature définit un ensemble d'instruments sur une même partie de la musique. Nous ne traitons pas les cas spéciaux ou des time signature peuvent coexister entre les instruments sur une même temporalité. Cet aspect permet de facilement définir une time signature et d'en changer en fonction du déroulé de la musique.

La représentation des concepts musicaux dans le DSL ont été décidé pour correspondre au langage du musicien. (Note, Track, TrackPart, time signature ...). Nous avons jugé notre dsl suffisament claire pour qu'un musicien comprenne les différents blocs.

5.1.1 Analyse de la pertinence et de l'efficacité :

Notre DSL est flexible de part la possibilité de spécifier le tempo, la time signature, et la réutilisation de TrackPart, offrant une représentation riche des structures musicales.

Les premier éléments que le musicien définit pour un son c'est la time signature et le tempo. Pour cette raison, nous avons décidé d'introduire le tempo au début du son, de plus nous lui donnons la possibilité de le changer dans un Track bien spécifique. La notion de TrackSet permet d'avoir des pistes sous une time signature commune, ce choix facilite l'utilisation de notre DSL de sorte que le musicien n'a pas besoin de répéter la time signature à chaque piste. Par conséquent, il définit sa time signature et se concentre seulement sur la déclaration des notes.

La gestion des erreurs humaines est prise en compte avec l'attribut "human error" dans la classe Track, offrant la possibilité d'injecter des erreurs au niveau de la piste. De la sorte, nous rendons la sensation d'écoute plus organique qu'avec une parfaite synchronicité des notes faite par ordinateur. Nous offrons la possibilité d'injecter l'erreur humaine même dans une note bien précise. Ce choix permet au musicien de personnaliser l'erreur humaine entre les Track. Par exemple, si l'utilisateur veut jouer des notes avec erreur dans une piste et d'autre dans une autre piste cela est réalisable facilement avec notre DSL.

La mise en place de TrackSet et la possibilité de réutiliser des TrackPart via ReuseTrackPart contribuent à la modularité et à la réutilisabilité des compositions musicales. Ce qui offre à l'artiste du confort dans la réutilisation de ses refrains par exemple. De plus, on donne la possibilité de réutiliser des track part tout en modifiant certaines notes. Donc, malgré des variantes entre refrain, l'artiste gagne en confort car il ne réécrit pas tout s'il doit changer quelque note de son refrain.

5.1.2 Les forces de notre DSL

La principale force de notre DSL réside dans la réutilisation et la répetition d'une partie de la musique (trackPart).

Nous pouvons répéter un TrackPart sur plusieurs mesures facilement. L'attribut repeat du TrackPart permet de réaliser cette tâche en passant juste le nombre de répétitions (qui correspond au nombre de mesures sur

lesquelles on veut copier les notes).

Nous offrons au musicien de jouer plusieurs instruments depuis son clavier. Il peut faire le mapping entre les touches du clavier et les notes, c'est un point positif puisque le musicien peut changer la configuration dans chaque scénario. L'utilisateur peut ensuite générer le fichier de scénario .music correspondant qui lui détaille les notes, leurs durées, volumes et temps de début. Il peut récupérer ce fichier et le modifier en ajoutant ou supprimant des notes.

Pour la fonctionnalité de musique intéractive, nous offrons pas la possibilité de jouer plusieurs instruments. L'utilisateur doit choisir un instrument pour jouer du son. Nous générons le fichier ".music" correspondant au son joué à partir du clavier. Cela permet à l'utilisateur d'enregistrer une partie avec son clavier et de pouvoir la modifier plus tard ou la fusionner dans une autre piste. Ceci permet à l'utilisateur d'écrire plus facilement de la musique.

5.1.3 Limitations ou défis rencontrés

Une des limitations de notre langage est que l'utilisateur ne peut pas jouer plusieurs instrument en même temps avec la fonctionnalité de musique interactive. En effet, on aurait aimé pouvoir mapper certaines touches du clavier à des instruments de piano et d'autre à une guitar par exemple, mais notre langage ne le permet pas. Il devra donc jouer ses deux instruments séparément et faire la fusion en ensuite.

Il n'est pas possible d'avoir simultanément deux signatures rythmiques différentes. Lorsque vous définissez un ensemble de pistes, la signature rythmique s'applique à tous les instruments de cet ensemble. Par conséquent, il n'est pas envisageable d'avoir deux instruments jouant en même temps avec des signatures rythmiques différentes. Nous avons jugé que cette situation est assez rare pour ne pas nécessiter une prise en compte spécifique. Cependant, si un utilisateur exprime le besoin, une mise à jour pourrait être envisagée. Il n'y aurait pas besoin d'une refonte trop conséquente.

Nous avons rencontré deux grands défis dans l'élaboration de ce DSL :

Apprentissage de la musique : La musique, en tant que discipline complexe, possède son propre langage, ses expressions, ses concepts mathématiques, son argot et ses règles. Au sein de l'équipe, aucun membre n'avait de connaissances en solfège au départ, ce qui constituait une barrière significative pour comprendre les nuances du domaine. Sans cette base, il aurait été difficile de saisir les attentes du client de manière adéquate.

Choix des technologies : Une fois le domaine musical appréhendé, nous avons entrepris un Proof of Concept (POC) pour garantir notre capacité à développer un Domain-Specific Language (DSL) sans être restreints par les limites des bibliothèques initialement choisies. Nous avons effectué deux changements de technologies, passant de JavaScript à Python pour le code généré. Au fur et à mesure de l'enrichissement de notre grammaire et de la complexification de notre DSL, nous avons pris conscience des limites des bibliothèques initialement sélectionnées.

5.2 Justifications de nos choix technologiques

5.2.1 Choix du DSL interne et Langium

Opter pour la création d'un DSL externe destiné aux musiciens se révèle être une décision judicieuse. Cette approche prend en compte le fait que les musiciens ne sont pas nécessairement versés dans les aspects techniques, soulignant ainsi l'importance de leur offrir une syntaxe accessible et intuitive. L'utilisation de notre DSL ne requiert pas d'expérience préalable en programmation. Nous avons spécifiquement conçu notre DSL pour qu'il soit aligné sur le langage et les termes habituellement employés par les professionnels de la musique.

L'avantage d'adopter un DSL externe est également sa capacité à opérer de manière indépendante par rapport à tout langage de programmation spécifique. Cette indépendance nous confère une liberté totale en termes de contrôle sur l'expressivité et la syntaxe de notre DSL, nous permettant ainsi de l'adapter au mieux aux besoins spécifiques de notre domaine d'expertise musical.

D'autre part, ce choix nous rajoute de la complexité pour maintenir l'intérpreteur de notre DSL. Nous utilisons également une librairie externe pour générer du son, donc il faut la maintenir également pour garantir la continuité du fonctionnement du DSL.

Pour réaliser ce DSL externe nous avons choisi d'utiliser la technologie Langium. Les raisons de ce choix sont les suivantes :

1. La facilité de définition de la grammaire, en effet langium offre un définition très proche du format BNF. Cette simplicité permet d'avoir une grammaire plus naturelle et compréhensible.
2. Génération facile de l'AST (abstract syntax tree) : langium permet de générer une abstraction de la grammaire en une seule commande. Cet AST contient également des fonctions auxiliaires pour parcourir les éléments de l'AST (vérification des types, ...). Cela nous rajoute de la facilité pour créer notre DSL.
3. CLI facilement utilisable : Pour appliquer la grammaire, on lance la commande "npm run langium :generate", et pour compiler notre générateur on lance la commande "npm run build". De plus, la CLI est configurable pour ajouter des validateurs, interpreteurs, etc. C'est un avantage pour enrichir notre DSL dans le futur.
4. Le langage du générateur : le langage est utilisé est typescript, il est facile à utiliser. Comme tous les membres de l'équipe sont à l'aise avec ce langage.
5. Extension langium sur vscode : la présence de cette extension nous facilite le développement. L'IDE est capable de localiser les erreurs de syntaxe. De plus, la documentation de langium est bien détaillée .
6. L'intégration avec l'Écosystème Language Server Protocol (LSP) : Langium supporte LSP, ce qui permet d'intégrer facilement les DSLs créés avec des éditeurs de texte et des environnements de développement intégrés (IDE) populaires. Cela offre une expérience utilisateur riche avec des fonctionnalités telles que l'auto-complétion, la vérification syntaxique, et la navigation dans le code. C'est ce qui nous a permis de faire facilement notre extension vscode pour notre langage.

5.2.2 Choix de la librairie MIDIUtil (python)

Nous avons essayé plusieurs librairies qui permettent de générer du son vers le format mid. Nous avons choisi dans un premier temps la libraire midi-writer-js, cette dernière nous permettait de jouer les sons avec toutes les configurations de base comme le tempo, la time signature, la définition des notes, les instruments, etc.

Cependant, après l'intervention de l'expert du domaine nous avons ressenti le besoin de placer des notes dans le même temps/beat. Midi-writer-js n'offrait pas la possibilité d'avoir cette fonctionnalité, elle nous obligeait d'avoir une seule note à chaque temps. Par conséquent, nous avons décidé de changer cette librairie.

Le choix final est fait sur la librairie python MIDIUtil . Nous avons constaté qu'elle répondait parfaitement à notre besoin. La définition d'une note est entièrement configurable en précisant le pitch de la note, sa durée, son début et son volume. Elle permet de jouer également tous les instruments avec des chaînes différentes. Nous avons vérifié que toutes les fonctionnalités qu'on voulait implémenter était réalisable avec cette libraire.

6 Répartition des tâches

Nous avons travaillé équitablement sur l'ensemble des tâches et fonctionnalités. Voici comment nous ne sommes réparti les tâches sur ce projet :

Jules	- Définition de la grammaire - Création Add on autoplay -Modification du kernel
Hamza	- Définition de la grammaire - Extension de l'erreur humaine - Extension de la modification des Bars - Développement du Kernel du générateur - Human Error
David	- Définition de la grammaire - Développement du Kernel du générateur - Feature interative input - Extension vscode
Théophile	- Définition de la grammaire - Diagramme de classe - Développement et modification du Kernel du générateur