UNIVERSITÉ CÔTE D'AZUR / POLYTECH'NICE SOPHIA
M2 - SI5
2023-2024

Julien Deantoni / Luc Hogie

Domain Specific Language

MusicML

Project delivery

The project concerns the second delivery of the DSL course. It will include the implementation of a DSL focused on the specification of tabs for at least a drum, in either an internal or an external DSL. Deliveries are expected by email (to Julien Deantoni: firstname.name@univ-cotedazur.fr, with [DSL] as object prefix) followed by "Group X" where X is your group number. The delivery is expected before the 21^{st} of January 2024 10:00PM Paris Time. The delivery is expected as a PDF report (please let your report be succinct and rigorous).
The report must contain :

- the name of the members of your team
- a link to the code of your DSL (typically a link your code's repository)
- a description of the language proposed:
  - the domain model represented as a class diagram;
  - the concrete syntax represented in a BNF-like form;
  - a description of your language and how it was implemented;
- a set of relevant scenarios implemented by using your language(s) (internal or external);
- a critical analysis of (i) DSL implementation with respect to the MusicML use case and (ii) the technology you chose to achieve it;
- responsibility of each member in the group with respect to the delivered project.

Objectives: Define the MusicML language

Your objective here is to define a DSL allowing the description of a music piece, which we see as a set of tracks evolving on time. A track is defined as a time-constrained sequence of notes. Time is defined by time signatures, it is "organized" by a (possibly infinite) sequence of *bar*s. A bar is divided by a sequence of *n beat*s. In turn, each beat is sub-divided in a number of *ticks*. This number of ticks per beat defines the *resolution* of a track. Common values for the resolution are one quarter or one third of a beat. A position in time is hence defined by a $(bar, beat, tick)$ tuple. Notes have a position in time, a value and a duration. Possible values for a note are known in Latin countries as *do, do#, re, re# mi, fa, fa#, sol, sol#, la, la# si* (the very same notes are referred to as *C, C#, D, D# E, F, F#, G, G#, A, A#, B* in English-speaking

countries). The duration of a note is expressed as a number of ticks. The logical time defined hereinbefore is augmented by the notion of *tempo* which define its projection to physical time. The tempo is the number of beat per minute (BPM).
Drum track are a little specific: A drumkit is a musical instrument made up of different elements, each producing a specific timbre. These elements are known as:

- kick drum or bass drum (*bd*),
- snare drum (*sd*),
- closed hihat (*ch*),
- opened hihat (*oh*),
- crash cymbal (*cc*)
- ride cymbal (*rc*).

In this project, just like in the MIDI (Musical Instrument Digital Interface) `https://fr.wikipedia.org/wiki/Musical_Instrument_Digital_Interface` (which is the *de facto* standard for the encoding of "event-based" digital music), each drum element is assigned to a specific note. Check out this data sheet for the assignment of a complete percussion set: `https://drummagazine.com/drum-notation-guide-2/`.
From your programs in the MusicML language, you will generate executable code. The target language and used library is your choice but it is required to use MIDI.
In MIDI, you have access to several instruments (called *programs*) that can be assigned to channels. A song (or sequence) consists of 16 distinct channels. Channels play simultaneously, thereby enabling multi-track sequences. Commonly drums are assigned to channel 10. In the drum channel (in the channel to which a drum sound is assigned), notes trigger drum elements, as given by the following mapping: `https://en.wikipedia.org/wiki/File:GMStandardDrumMap.gif`, or here: `https://www.midi.org/specifications-old/item/gm-level-1-sound-set`).

Support for other instruments

not an option after discussion with the Domain Expert, sorry
Midi supports other instruments (`https://www.zikinf.com/articles/mao/tablemidi.php`). It may be interesting to provide support for other instruments, playing at the same time then the drum. In this case you should enable the description of the different parallel bar but also the duration of the notes, which does not hold for the drum; see here for an example of note duration: `https://www.songsterr.com/a/wsa/michael-jackson-billie-jean-tab-s10586t4`.

Basic scenario

1. **Billie Jean, Mickael Jackson** `https://youtu.be/V29FNfECL9k`: in this example we will also consider only the beginning of the song - the 34 first bars until the chorus comes in. The drum pattern uses 3 elements: closed hihat, snare drum and bass drum. It steadily repeats 4 beats forming one single bar. Bass drum is played on every two beats (*i.e.,* on beat 1 and 3). Snare drum emphasizes beats 2 and 4, and hihat hammers every quarter-note. Bass only plays 8th notes.

   `https://www.songsterr.com/a/wsa/michael-jackson-billie-jean-drum-tab-s10586t1`

   The DSL should enable the user to describe every variation in this song, including the drum fill introducing the chorus, and the tom-tom fills which augment the pattern in the verse after the chorus.

2. **Love Is All**: In this example, you will find two different time signatures. While most of the song is in 4/4; there is a bridge that runs in 3/4, with a tempo scaling up several times at different places. The tab can be found here:
   `https://www.songsterr.com/a/wsa/roger-glover-love-is-all-drum-tab-s52270`. The DSL should allow the specification and code generation for such cases.

Common Parts

The following parts must be available in your DSL:

- **Abstract syntax**:

  The abstract syntax should be clearly identified in the delivered code.

- **Concrete syntax**:

  The concrete syntax (external or embedded) must be clearly identified and used by a relevant set of scenarios. The syntax must leverage the tool chosen to implement it to make it clear and easy to use.

- **Validation**:

  Support your end-user by checking that a model is realizable on the expected platform. For example, choice of the instruments, tempo, etc.

- **Code generation**:

  Provide a generator producing turn-key midi code, playing the rhythm. This code can be directly compiled (if needed), and played. The target language is your choice but it required to use the Musical Instrument Digital Interface (`https://fr.wikipedia.org/wiki/Musical_Instrument_Digital_Interface`).

- **DSL Addon for User Acceptance**: The acceptance of a language by users is usually not only due to the language itself but also on the extra services provided by the languages. These services can be related to the language itself (like the *borrowchecker* in Rust) or used here to help the user in the definition of its programs (like for instance a graphical representation of the control flow or a very precise and didactic error spotting). You will provide at least one extra service for the MusicML DSL.

"À la carte" features

The remainder of this document describes some extensions that you can implement on top of your language. Each extension is defined by a short description and at least one acceptance scenario Choose (at least) one feature to introduce in your project.

~~Support for other instruments~~

~~Midi supports other instruments (`https://www.zikinf.com/articles/mao/tablemidi.php`). It may be interesting to provide support for other instruments, playing at the same time then the drum. In this case you should enable the description of the different parallel bar but also the duration of the notes, which does not hold for the drum; see here for an example of note duration: `https://www.songsterr.com/a/wsa/michael-ja`~~

Support for bar modifications

It is often the case that some bars are variations of a previously defined bar. It is consequently interesting to allow the user to define a bar based on a previous one. It could for instance specify that the new bar is equal to bar #n but with a open hihat at beat #3 instead of a close hihat. In this extension, consider different kinds of manipulation like addition of new notes, replacements, removing, etc.

Support for human like errors

Some drumers are known to have specific small timing errors that make the music more pleasant. Instead of trying to explain you, you'd better read this: https://news.harvard.edu/gazette/story/2012/07/when-the-beat-goes-off/ or this: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0026457. Choosing this extension means providing a way for your user to characterize the error and inject it in the generated code.

Support for user interactive input

By choosing this extension, you allow you user to define a mapping between keyboard input and a note of a specific instrument. When the resulting program is played, the user can play on the keyboard, conjointly with music as defined in its program.

Support for user defined sounds

There exist two ways to do music in Java: the Midi and the Audio API (It is usually the same for other languages).
Midi is based on soundbank, being a collection of instruments defined in a soundFont (https://en.wikipedia.org/wiki/SoundFont. Audio is based on the possibly parallel read of some wav files. It is then possible to play any kind of sounds so that a user could create its own sound samples and create tabs from them.
By choosing this extension, you allow the user to import its own samples, to modify them (add silence, concatenate, cut, etc). It must also be possible for the user to use its samples in the most appropriate way to create her/his own mix of samples. Note that this extension can be used in parallel with the MIDI part previously defined.