Technical Report: Graphical Music Compiler

Esteban Alejandro Villalba Delgadillo, Daniel Felipe Barrera Suarez

Universidad Distrital Francisco Jose de Caldas, Bogotá, Colombia

***Abstract*—This project presents the design of a graphical-musical interpretive system that transforms structured textual instructions into visual representations of musical notation. Inspired by the domain-specific language (DSL) paradigm, our approach implements a processing pipeline that parses commands such as `play [note] for [duration] at [time]`, builds an intermediate piano roll representation, and generates visualizations using `matplotlib`. We detail the basic grammar, the internal data model, and the note-to-graph mapping, establishing the foundation for future extensions including real-time sound synthesis via `pygame`.**

***Index Terms*—Domain-specific languages, computational music, sound visualization, DSL, Pygame, piano roll.**

I. Introduction

Domain-specific languages (DSLs) have proven their educational value by enabling complex concepts to be expressed through simplified syntax. In computer science, this paradigm is often explored by creating micro-languages that generate graphical outputs from minimalist text commands.

Our project applies this idea to the musical domain by developing an interpreter that converts commands such as:

```
play C4 for 1.0 at 0.0
play E4 for 0.5 at 1.0
```

into *piano roll* visual representations, a standard format in digital music production.

II. Literature Review

Languages such as Sonic Pi [1], FoxDot [2], and TidalCycles have shown that it is possible to manipulate music in real time through code. However, many of these require advanced technical knowledge. Our approach aims for pedagogical accessibility through simplified design.

Additionally, libraries like Pygame allow sound playback in a straightforward way, making it feasible to combine symbolic notation, sound reproduction, and graphical visualization in educational environments.

III. Background

The initial interpreter will use lists of instructions to trigger sounds using `pygame.mixer` or similar libraries.

This project is grounded in theoretical principles studied in Workshop 1 of the course, specifically finite automata, regular expressions, and context-free grammars. These concepts inform the design of the language's lexical and syntactic structure. The interpreter's ability to model rules for notes, durations, and repetitions stems directly from grammar modeling and derivation trees explored in class. Thus, the workshop provided the core foundation for building a functional and parsable DSL.

IV. Objectives

- Design a symbolic interpreter for musical phrases.
- Create a textual interface to interpret these phrases.
- Implement a sound execution prototype.
- Establish the foundation for real-time visualization.

V. Scope

Prototype limitations:

- Only simple musical notes.
- Basic durations (quarter, eighth notes).
- Sequential execution.
- Textual or *piano roll* visual representation (no traditional scores).

VI. Assumptions

- User has basic programming knowledge.
- Note format: `C4`, `D#5`.
- Fixed tempo (e.g., 120 BPM).
- No concurrency or parallel execution.

VII. Limitations

- No live editing or real-time execution.
- Monophonic playback.
- Sound quality limited to .wav files or basic tones.
- No MIDI export or graphical score generation at this stage.

VIII. Methodology

The methodology begins with the definition of a minimal symbolic language that allows users to describe notes and their durations. For instance, inputs such as `["C4-1/4", "E4-1/4", "G4-1/2"]` are parsed using a simple recursive descent parser. The interpreter, implemented in Python, uses `pygame.mixer` to play notes sequentially.

Once parsing is successful, a visualization is generated using `matplotlib`, where each note is represented as a bar on a piano roll diagram. Future work includes expanding the syntax to support chords, loops, and live manipulation.

IX. Glossary

- **BPM**: Beats Per Minute, measures the musical tempo.
- **Pygame**: Python library for multimedia development.
- **Musical Syntax**: Textual representation of notes and durations.

Acknowledgements

We thank the course team for their guidance in technical design and the academic institution that encourages projects combining music and technology.

Appendix (Future Work)

The appendix will include syntax examples, code snippets, and the folder structure of the project. It will be added once implementation progresses further.

References

[1] Sonic Pi. https://sonic-pi.net
[2] FoxDot. https://foxdot.org
[3] Brown, C. et al. "Designing DSLs for Education", *Proc. ACM Educators Workshop*, 2019.
[4] Dannenberg, R. "Music Representation Issues, Techniques, and Systems", *Computer Music Journal*, 2021.
[5] Pygame Documentation, https://www.pygame.org/docs