# Technical Report: Graphical Music Compiler

Esteban Alejandro Villalba Delgadillo, Daniel Felipe Barrera Suarez

Universidad Distrital Francisco Jose de Caldas, Bogotá, Colombia

***Abstract*—This project proposes 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 analyzes commands such as `play [note] for [duration] at [time]`, builds an intermediate representation in *piano roll* format, and generates visualizations using *matplotlib*. The document outlines the basic grammar, the data structure of the *piano roll*, and the note-to-graph mapping process, laying the groundwork for future extensions with sound synthesis.**

***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, FoxDot, 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

A musical language requires:

- Symbolic notation: notes like `C4`, `G#3`, durations like `1/4`, `1/8`.
- An execution engine to play the notes.
- Grouping of musical events into phrases/patterns.

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

This project is grounded in theoretical principles in the domains of finite automata, regular expressions, and context-free grammars. These foundational topics inform the design of the interpreter's syntax and parsing logic. By understanding how to model real numbers, identifiers, and structured expressions through formal grammars and derivation trees, we ensure that our compiler remains both expressive and parsable. The interpreter's language definition and execution flow directly benefit from this theoretical framework, making the workshop's insights crucial to the success of this implementation.

## 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

- **Language Definition**: Conventions for notes, silences, repetitions.
  Example: `["C4-1/4", "E4-1/4", "G4-1/2"]`
- **Interpreter Construction**: Python script to play sequences using `pygame.mixer`.
- **Testing**: Simple combinations of notes/durations.
- **Future Extensions**: Support for loops, polyphony, visualizations.

## 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.


## Appendices

(to be added in future versions)

- Syntax examples
- Python code snippets
- Project folder structure

## References


[1] Brown, C. et al. "Designing DSLs for Education", *Proc. ACM Educators Workshop*, 2019.

[2] Dannenberg, R. "Music Representation Issues, Techniques, and Systems", *Computer Music Journal*, 2021.

[3] Pygame Documentation, https://www.pygame.org/docs