WESTERN MUSIC NOTATION FOR JAVA: A LIBRARY FOR MUSIC NOTATION ON THE JVM

Otso Björklund
University of Helsinki
otso.bjorklund@helsinki.fi

ABSTRACT

This paper presents the wmn4j Java library for handling Western music notation. The central goal of wmn4j is to provide a simple model of musical scores and an intuitive API that allows efficient access to their contents. Wmn4j supports fully concurrent and parallel access to all contents of scores and is intended for implementing large scale server-side music analysis applications. Wmn4j is licensed under the MIT license and is available on Github [1] and Maven Central [2].

1. INTRODUCTION

Wmn4j is a Java library for analysing and producing western music notation. The library is focused especially on computational analysis of symbolically represented music. Wmn4j is available on Maven central and can be easily installed as a dependency using the package managers of different JVM languages. The central goal of wmn4j is to offer efficient access to the contents of musical scores. The main use cases for wmn4j are large scale corpus analyses and server applications for computational music analysis.

Python libraries such as music21 [1] and partitura [3] offer functionality for analysing musical scores. While wmn4j has similar use cases as music21, there are some crucial differences. Wmn4j targets the Java Virtual Machine (JVM) environment, which enables wmn4j to be easily used in multiple languages targeting the JVM. The design of wmn4j makes it especially suitable for use in functionally oriented modern JVM languages, such as Clojure [4] and Scala [5]. Music21 offers a wide range of algorithms and tools for music analysis. The scope of wmn4j is narrower, as it mainly aims to provide access to the contents of scores and contains only few music analysis algorithm implementations. Partitura is focused on modeling musical expression in addition to working with symbolic musical data. Wmn4j is entirely focused on the contents of music notation and does not aim to support modeling musical expression.

The above-mentioned Python packages for handling music notation provide convenient tools for research in MIR and computational musicology. Wmn4j can be used for implementing experimental research code, however, its main goal is to provide a robust and efficient library for implementing large-scale applications dealing with music notation as data.

2. FEATURES

Wmn4j currently targets Java 17, the latest long-term support version, and is designed and implemented with modern Java in mind. Wmn4j doesn't aim to provide a comprehensive collection of music analysis or information retrieval algorithms. Instead, it aims to provide classes and an API that make efficient implementation of algorithms easy. Currently wmn4j offers a basic pattern matching algorithm implementation based on [2] which enables finding exact matches of a pattern from polyphonic music. In order to make wmn4j suitable for larger applications, logging is implemented using SLF4J [6] which allows application developers to use any SLF4J compatible logger implementation with wmn4j.

2.1 The structure of scores in wmn4j

The structure of a score in wmn4j is modelled directly as a containment hierarchy depicted in Figure 1. The objects in musical notation are mostly mapped directly to classes that represent them. The notation elements contained within a measure that have a duration implement the Durational interface. This is based on the use of DurationalSymbol in the class hierarchy presented in [3]. The API of wmn4j is comprehensively documented [7].

The classes that model the contents of music notation are immutable, making them inherently thread-safe [4]. All contents of musical scores can thus be accessed concurrently and in parallel without requiring any additional synchronization. The classes also guarantee strong invariant conditions, such as durations always having positive

[1] https://github.com/otsob/wmn4j
[2] https://search.maven.org/artifact/org.wmn4j/wmn4j
[3] https://github.com/CPJKU/partitura
[4] https://clojure.org
[5] https://scala-lang.org
[6] https://www.slf4j.org
[7] https://otsob.github.io/wmn4j/

Late-Breaking / Demo Session Extended Abstract, ISMIR 2022 Conference

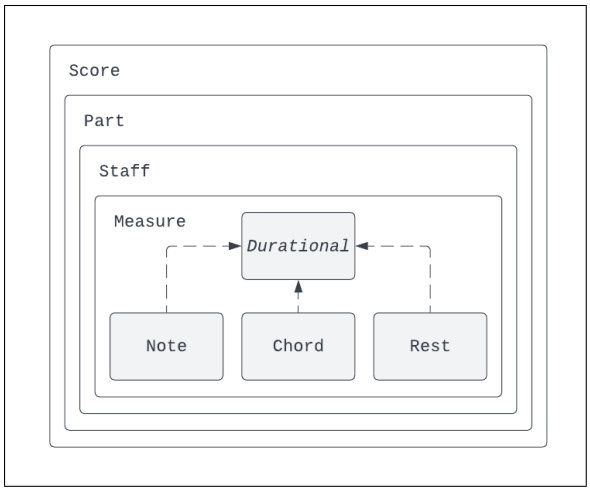

Figure 1. `Wmn4j` score containment hierarchy

values representable as a rational number. All methods in the score API of `wmn4j` that may return an empty value explicitly use Java's `Optional` type. Builder classes are provided for making programmatic creation of scores easier. The package structure of `wmn4j` has been designed so that it can be further split into Java modules if the project grows larger and only some parts of it need to be used on platforms with resource constraints.

2.2 Accessing notation elements

Scores are modelled in `wmn4j` using a simple containment hierarchy that resembles the tree-structure of MusicXML. Representing music as a single hierarchy may not be ideal for all computational music analysis tasks [5]. The approach taken in `wmn4j` is to decouple the structure used to store scores in memory from methods of accessing the contents. As the data structures used to model a score are immutable, multiple different *views* can be built on top of the same instance of a score without having a need for performing defensive copying to avoid unintented side effects.

The main abstractions `wmn4j` uses for providing access to the contents of scores are iterators, streams [8], *positions*, and *selections*. The iterators and streams follow the requirements of the corresponding Java interfaces they implement. Each `Durational` notation element has a unique position in a score. This is modelled by the `Position` class, which can also be used to access the element at a specific position. The `Selection` class provides a way to select a range of measures and specific parts of a score. Using the abstractions `wmn4j` provides for accessing the contents of a score, the strictly hierarchical structure of the `Score` class does not constrain how content can be accessed.

2.3 MusicXML input and output

The only file format `wmn4j` currently supports is MusicXML [6]. MusicXML files can both be read and written, however, the current version of `wmn4j` does not have support for all data MusicXML files can contain. The MusicXML reader in `wmn4j` uses Java's streaming XML API, which enables highly efficient XML input. Even even large symphonic scores can be efficiently read with a relatively small memory footprint.

3. USE CASES

The main use cases of `wmn4j` are in implementing applications that need to analyze large corpora of music. Multiple big data frameworks, such as Apache Spark [9] target the JVM and can be extended using Java. `Wmn4j` could thus be used for implementing large-scale musical score analysis on top of an existing big data framework. Example code for running multithreaded analyses on scores with `wmn4j` is available on GitHub [10].

The GraalVM Native Image builder [11] can be used to compile JVM byte-code into native code in order to avoid the start-up time of the JVM. `Wmn4j` supports native build with little need for additional configuration, and can be used also for developing command line applications and serverless microservices that require faster startup. The `mncmd` [12] application illustrates how `wmn4j` can be used with Clojure and GraalVM's Native Image builder to implement a command line application for extracting information from MusicXML files.

4. FUTURE WORK AND CONCLUSION

The API and feature set of `wmn4j` is not fully stable yet, and work on making all information contained within MusicXML available in `wmn4j` will be continued. Support for input and output of MEI [7] may potentially be added in the future. In order to stabilize `wmn4j`'s current API, feedback from actual application development use is still required. Until then, `wmn4j` will be versioned under major version zero in accordance with semantic versioning practices.

The various software libraries for operating on music notation data provide valuable tools for MIR research and computational musicology. `Music21` and `partitura` already cover the needs of researchers well in many areas. The main contribution of `wmn4j` is providing an efficient software library with a clear API for developers aiming to implement large-scale music analysis applications.

5. ACKNOWLEDGEMENTS

The author would like to thank Matias Wargelin for his significant contributions to the development of `wmn4j`.

[8] `https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/stream/Stream.html`

[9] `https://spark.apache.org`

[10] `https://github.com/otsob/wmn4j-ismir22-lbd`

[11] `https://www.graalvm.org/22.1/reference-manual/native-image/`

[12] `https://github.com/otsob/mncmd`

6. REFERENCES

[1] M. S. Cuthbert and C. Ariza, "Music21: A Toolkit for Computer-Aided Musicology and Symbolic Music Data," in *Proceedings of the 11th International Society for Music Information Retrieval Conference*, Utrecht, Netherlands, 2010, pp. 637–642.

[2] E. Ukkonen, K. Lemström, and V. Mäkinen, "Geometric Algorithms for Transposition Invariant Content-Based Music Retrieval," in *Proceedings of the 4th International Conference on Music Information Retrieval*, Baltimore, Maryland, USA, 2003.

[3] K. Lassfolk, "Music Notation as Objects. An Object-Oriented Analysis of the Common Western Music Notation System," Dissertation, University of Helsinki, 2004.

[4] J. Bloch, *Effective Java*™, 2nd ed. USA: Prentice Hall Press, 2008.

[5] G. A. Wiggins, "Computer-Representation of Music in the Research Environment," in *Modern Methods for Musicology: Prospects, Proposals, and Realities*, T. Crawford and L. Gibson, Eds. Ashgate, 2009, pp. 7–22.

[6] M. Good, "MusicXML: An Internet-Friendly Format for Sheet Music," in *Proceedings of XML*, Boston, United States, 2001.

[7] P. Roland, "The Music Encoding Initiative (MEI)," in *MAX2002. Proceedings of the First International Conference on Musical Application using XML*, 2002, p. 55–59.