Abjad: A Python API for Formalized Score Control

Trevor Bača
Harvard University
trevor.baca@gmail.com

Josiah Wolf Oberholtzer
Harvard University
josiah.oberholtzer@gmail.com

Jeffrey Treviño
Carleton College
jeffrey.trevino@gmail.com

Víctor Adán
vctradn@gmail.com

ABSTRACT

Place your abstract at the top left column on the first page. Please write about 150-200 words that specifically highlight the purpose of your work, its context, and provide a brief synopsis of your results. Avoid equations in this part.

1. INTRODUCTION

Abjad [1] is an interactive software system designed to help composers build up complex pieces of music notation in an iterative and incremental way. Abjad is implemented in the Python [2] programming language and architected as an interrelated collection of packages, classes and functions. Users can visualize their work as publication-quality score at all stages of the compositional process using Abjad's interface to the LilyPond [3] music notation package. Abjad is open-source software available for free download from the Python Package Index [4].

2. BACKGROUND & HISTORY

2.1 Composition as Notation

Conventional western notation implies two kinds of information analysis: graphic specification divides an instruction into perceptually fused but independently specified aspects (pitch, rhythm, dynamic) and segregates event information (notes) from sound production information (instructions for how to play an instrument to produce sound, described via performance practice or graphically through tablature notation) [1, 83]. While many environments for both notation and sound production have arisen within the last twenty years, the following discussion focuses solely on the production of notation: Abjad enables composers to express both low- and high-level compositional ideas with the aid of a widely used programming language that provides a sufficiently detailed object model of common practice musical notation. The present discussion defines composition narrowly as the act of creating a document.

[1] www.projectabjad.org
[2] www.python.org
[3] www.lilypond.org
[4] pypi.python.org

2.2 Computational Models of Music/Composition

Computational models of music might entail the representation of higher-level musical entities apparent in the acts of listening and analysis but absent in the symbols of notation themselves, as determined to be creatively exigent. Programming researchers and musical artists have modeled many such extrasymbolic musical entities, such as large-scale form and transition [2], [3], [4], [5], [6], texture [7], contrapuntal relationships [8], [9], [10], [11], [12], [3], [13], [14], [15], [16], [17], [18], harmonic tension and resolution [19], [20], [21], melody [22], [23], meter [24], rhythm [25], [26], [27], timbre [28], [29], [30], temperament [31], [32], and ornamentation [33], [34]. This work overlaps fruitfully with analysis tasks, and models of listening and cognition can enable novel methods of high-level musical structures and transformations, like dramatic direction, tension, and transition between sections [35, 108].

The set of typographical notation symbols overlaps with the set of extra-symbolic musical entities, often causing confusion in discourse and program architecture. For example, a "chord" might be a vertically ordered collection of pitch classes in a harmonic conceit, or it might refer to the specific arrangement of pitched noteheads, stemmed together into a composite notation symbol that instructs a performer to perform a sound that consists of several component pitches. Due to substantial overlap in vocabulary between musical and notational concepts, it can be difficult to separate a system's model of music/composition from its model of notation.

A system that affords a detailed model of music/composition without linking it to a sufficiently detailed model of musical notation does not afford automated notation — sufficiency, however, depends heavily on generative task. For example, if a composer requires an automated notation system to render complex rhythmic ideas that depend typographically on nested tuplets, a system that produces a notation only via a combination of MIDI and quantization must reduce rhythms to a non-hierarchical stream of event times, eliminating the temporally divisive approach of tuplet notation. For many rhythmic applications, though, MIDI suffices.

2.3 Computational Models of Notation

Many automated notation systems exist to model musical notation and the act of typographical layout without explicitly affording the computational modeling of music or composition [36], [37], [38], [39]; many of these systems strongly imply a model of music, such as Grégoire for Gregorian chant, Django for guitar tablature, and GOODFEEL for Braille notation [40]. In this light, feature-rich systems oriented toward classical composers, such as Finale, Sibelius, SCORE, Igor, Berlioz, and Nightingale fit into the mold of systems that model notation with genre as a primary determinant of generative task. Such a system might go so far as to enable a text-based object-oriented model of notation that automates some aspect of an otherwise point-and-click interface, as in the case of Sibelius's ManuScript scripting language [41].

Many models of musical notation were created for purposes of corpus-based computational musicology. Formats such as DARM, SMDL, HumDrum, and MuseData model notation with the generative task of searching through a large amount of data [42]. Commercial notation software developers attempted to establish a data interchange standard for optical score recognition (NIFF) [43]; since its release in 2004, MusicXML has become a valid interchange format for over 160 applications and maintains a relatively application-agnostic status, as it was designed with the generative task of acting as an interchange format between variously tasked systems [44].

Notation representations that underly many of these GUI-based systems often go undescribed as computer representations of notation, in favor of discussions about human-computer interaction. For example, Barker and Cantor developed an early model of music notation that underlies a four-oscilloscope GUI and describe their work entirely in terms of user interaction [45]; likewise, discussions of modern commercial notation systems remain similarly oriented, without much awareness or criticism of the underlying computational models of notation. This results in insufficiently detailed models of notation; systems, for example, that provide models only of mensural notation and enable nonmensural notations only as modified instantiations of notations based on measures.

2.4 The Development of Hierarchical Object Models of Notation

Many early models of musical notation were not hierarchical, and Lejaran Hiller, in reflecting on decades of automated notation work, identifies the lack of hierarchical organization as a limitation of early work — although Nick Collins points out that even Hiller's program PHRASE addresses the hierarchical organization of a score up to the level of a phrase, without moving beyond this mid-level musical structure to concerns of large-scale form [35, 108].

There were several object-oriented music environments by 1990 [46, 139], most created in or inspired by the newly popular Smalltalk-80 programming language; while they facilitated the hierarchical modeling of musical abstractions, they omitted or radically simplified the hierarchical nature of common notation. For example, Glen Krasner (Xerox Systems Science Laboratory) created Machine Tongues VIII, a music system that created an object-oriented model of the score/orchestra distinction inherited from Max Mathews' Music N languages, with a simple linear model of "partOn" and "partOff" command sequences [47], omitting hierarchical organization entirely when the system produced notational output; although subsequent Machine Tongues systems introduced some hierarchical organization via "note" objects that inhabited "event lists," systems did not attempt to model the hierarchical detail of all a traditional score's elements. Like Hiller's PHRASE program, Andreas Mahling's CompAss system organized events hierarchically up to the mid-level "phrase" level of musical structure [48]. These systems extend Smalltalk-80 with interfaces to the MIDI communications protocol: as extensions of Smalltalk, they enabled the user to arbitrarily extend the system with new objects, creating a detailed and hierarchical model of music, usually flattened into a list of noteOn and noteOff commands to be notated or played back via MIDI interface.

By 1989, Glendon Diener's Nutation system (written in Objective C for the NeXT computer) modeled both musical and notational structure hierarchically through the use of directed graphs [49], [50], [51].

3. NOTATIONAL ISOMORPHISM

Abjad models objects on the page according to common practice notation.

3.1 Notational aggregation

We assume notational primitives are the elements of composition.

We consider the act of composition then to revolve around the iterative aggregation of notational primitives into arbitrarily complex score objects.

- append(), extend(), insert()
- attach()

3.2 Notational visualization

Abjad makes visualizing notational artifacts simple.

Any notational element or aggregate can be displayed at any time as a PDF via calls to its top-level `show()` function.

3.3 Explicit notational modeling

Abjad models notation explicitly. All notational primitives expressed by Abjad must conform to the principles of common practice notation. When compositional inputs cannot be expressed in terms of these principles, we provide affordances for massaging them into valid notational states.

For example, Abjad expresses the durations of all score components in terms of rational values – fractions and integers – rather than floating point numbers. Likewise Abjad expresses all pitches in terms of triples of diatonic note names, accidentals and octave numbers, rather than MIDI numbers or frequencies. While we provide alternative representations of pitch and rhythm, as well as affordances for moving between them, the format actually stored in and used by score components for rendering notation is always the most notationally-explicit.

3.4 Written, assignable and prolated durations

All `Note`, `Chord` and `Rest` objects in Abjad must be instantiated with a duration corresponding to the written glyphs on the page – a *written* duration.

Written durations must be *assignable*, a category we invented to model durational initialization. Durational assignability describes whether a duration can be represented as a power-of-two flag count combined with zero or more dots. `1/4`, `3/16` and `7/16` are assignable durations while `5/32`, `9/8` and `1/12` are not.

Non-assignable durations cannot be represented in common practice notation by a single glyph. They require two or more glyphs with assignable durations tied together, for the score component to be tupletted, or both.

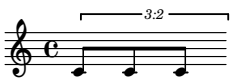
Abjad will not automatically render a single note with a duration of `5/16` as two or more notes tied together. We consider such behavior to be too implicit. There are too many potentially compositionally valid ways to render a duration such as `5/16` into a series of tied assignable durations: `1/4 + 1/16`, `3/16 + 2/16`, `2/16 + 3/16`, `1/16 + 1/4`, `1/8 + 1/8 + 1/16` etc. Instead we provide affordances for generating tied notes from non-assignable durations. One such affordance is our `scoretools.make_notes()` function, which chooses smart defaults for generating tied glyphs from otherwise un-notateable input.

```
>>> staff = Staff(selection)
>>> show(staff)
```

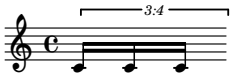
All score components also have a *prolated* duration - the product of their written duration and their *prolation*. Prolation is the cumulative product of all the *multiplier* of every tuplet found in the *parentage* of a score component. A score component's prolation depends on its location in the score hierarchy, and is not an inherent property of itself independent that hierarchy.

Three `Note` objects each having a prolated duration of `1/12` can be represented as either three `1/16` notes in a `3:4` tuplet or as three `1/8` notes in a `3:2` tuplet. As all Abjad `Note` objects must have an assignable written duration, the three notes above must have written durations of either `1/8` or `1/16`, and the tuplet must be correspondingly an explicit diminution or augmentation to provide the desired prolation of `2/3` or `4/3`.

```
>>> tuplet = selection[0]
>>> show(tuplet)
```




```
>>> show(tuplet)
```



The durational information of any aggregate score object in Abjad is therefore always explicit and unambiguous with regard to its notational reality.

4. RELATIONSHIP MODELING

5. SCORE ADDRESSABILITY

6. EXTENSIBILITY

6.1 Compositional Agnosticism

By providing core functionality oriented toward the elements of standard notation, Abjad strives to remain as agnostic as possible to various composition techniques.

6.2 Example Extensions

To better afford high-level, personal, eccentric composition techniques as optional tools packages, Abjad provides clear examples of extensibility through the authors' own included extensions, including: - labeltools - metertools - quantizationtools - rhythmmakertools - selectortools - sievetools - tonalanalysistools

6.3 Affording Extension through Project Structure and Interface

As an open-source project, composers and researchers can contribute changes via git pull requests. A process of continuous integration and online version control simplifies this contribution process.

7. EMBEDDABILITY

Abjad is an importable Python library. It can be used in whole or in part as a component of any Python-compatible system. Abjad has few Python package dependencies and is not bound to any specific user application or graphic user interface. These qualities make Abjad an ideal project ideal for embedding in other software systems.

For example, Abjad supports IPython Notebook [5] , a web-based interactive computational environment combining code execution, text, mathematics, plots and rich media into a single document. Notational output from Abjad can be transparently captured and embedded directly into an IPython Notebook which has loaded Abjad's IPython Notebook extension.

8. OPEN SOURCE

9. REFERENCES

[1] C. Ariza, *An Open Design for Computer-Aided Algorithmic Composition: athenacl*. Dissertation. com, 2005, vol. 54. [Online]. Available: http://books.google.com/books?hl=en&lr=&id=XukW-mq76mcC&oi=fnd&pg=PR3&dq=An+Open+Design+for+Computer-Aided+Algorithmic+Composition:+athenacl&ots=bHedXym8ZP&sig=9i2RQINqIVr2Y7sjxeD9e74myxA

[2] L. Polansky, M. McKinney, and B. E.-A. M. Studio, "Morphological Mutation Functions," in *Proceedings of the International Computer Music Conference*, 1991, pp. 234–41.

[5] http://ipython.org/notebook.html

[3] Y. Uno and R. Huebscher, "Temporal-Gestalt Segmentation-Extensions for Compound Monophonic and Simple Polyphonic Musical Contexts: Application to Works by Cage, Boulez, Babbitt, Xenakis and Ligeti," in *Proceedings of the International Computer Music Conference*, 1994, p. 7.

[4] C. Dobrian, "Algorithmic Generation of Temporal Forms: Hierarchical Organization of Stasis and Transition," in *Proceedings of the International Computer Music Conference*, 1995.

[5] S. Abrams, D. V. Oppenheim, D. Pazel, J. Wright *et al.*, "Higher-level Composition Control in Music Sketcher: Modifiers and Smart Harmony," in *Proceedings of the International Computer Music Conference*, 1999.

[6] M.-J. Yoo and I.-K. Lee, "Musical Tension Curves and its Applications," *Proceedings of International Computer Music Conference*, 2006.

[7] S. Horenstein, "Understanding Supersaturation : A Musical Phenomenon Affecting Perceived Time," *Proceedings of International Computer Music Conference*, 2004.

[8] G. Boenn, M. Brain, M. De Vos, and et. al., "Anton: Composing Logic and Logic Composing," in *Logic Programming and Nonmonotonic Reasoning*. Springer, 2009, pp. 542–547.

[9] A. Acevedo, "Fugue Composition with Counterpoint Melody Generation Using Genetic Algorithms," in *Computer music modeling and retrieval: Second International Symposium, CMMR 2004, Esbjerg, Denmark, May 26-29, 2004: revised papers*. Springer-Verlag New York Inc, 2005, p. 96. [Online]. Available: http://books.google.com/books?hl=en&lr=&id=zXegsi7tj00C&oi=fnd&pg=PA96&dq=Fugue+Composition+with+Counterpoint+Melody+Generation+Using+Genetic+Algorithms&ots=Z4DpBjPMN-&sig=ocgSVVHiDIB7GJfmznh1Z3OheUA

[10] T. Anders and E. R. Miranda, "Constraint Programming Systems for Modeling Music Theories and Composition," *ACM Computing Surveys*, vol. 43, no. 4, pp. 30:1–30:38, Oct. 2011. [Online]. Available: http://doi.acm.org/10.1145/1978802.1978809

[11] K. Balser and B. Streisberg, "Counterpoint Compositions in Non-tempered Systems: Theory and Algorithms," *Proceedings of International Computer Music Conference*, 1990.

[12] D. E. Jones, "A Computational Composer's Assistant for Atonal Counterpoint," *Computer Music Journal*, vol. 24, no. 4, pp. pp. 33–43, 2000. [Online]. Available: http://www.jstor.org/stable/3681553

[13] M. E. Bell, "A MAX Counterpoint Generator for Simulating Stylistic Traits of Stravinsky, Bartok, and Other Composers," *Proceedings of International Computer Music Conference*, 1995.

[14] M. Farbood and B. Schoner, "Analysis and Synthesis of Palestrina-style Counterpoint using Markov Chains," in *Proceedings of the International Computer Music Conference*, 2001, pp. 471–474.

[15] D. Cope, "Computer Analysis and Computation Using Atonal Voice-Leading Techniques," *Perspectives of New Music*, vol. 40, no. 1, pp. 121–146, 2002. [Online]. Available: http://www.jstor.org/stable/833550

[16] M. Laurson and M. Kuuskankare, "Extensible Constraint Syntax Through Score Accessors," in *Journées d'Informatique Musicale*, 2005, pp. 27–32.

[17] L. Polansky, A. Barnett, and M. Winter, "A Few More Words About James Tenney: Dissonant Counterpoint and Statistical Feedback," *Journal of Mathematics and Music*, vol. 5, no. 2, pp. 63–82, 2011.

[18] K. Ebcioglu, "Computer Counterpoint," *Proceedings of International Computer Music Conference*, 1980.

[19] A. F. Melo and G. Wiggins, "A Connectionist Approach to Driving Chord Progressions Using Tension," in *Proceedings of the AISB*, vol. 3, no. 1988, 2003. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.115.9086&rep=rep1&type=pdf

[20] G. Wiggins, "Automated Generation of Musical Harmony: What's Missing?" in *Proceedings of the International Joint Conference on Artificial Intelligence*, 1999. [Online]. Available: http://www.doc.gold.ac.uk/~mas02gw/papers/IJCAI99b.pdf

[21] C. D. Foster, "A Consonance Dissonance Algorithm for Intervals," *Proceedings of International Computer Music Conference*, 1995.

[22] D. Hornel, "SYSTHEMA - Analysis and Automatic Synthesis of Classical Themes," *Proceedings of International Computer Music Conference*, 1993.

[23] M. Smith and S. Holland, "An AI Tool for Analysis and Generation of Melodies," *Proceedings of International Computer Music Conference*, 1992.

[24] M. Hamanaka, K. Hirata, and S. Tojo, "Automatic Generation of Metrical Structure Based on GTTM," *Proceedings of International Computer Music Conference*, 2005.

[25] P. Nauert, "Division- and Addition-based Models of Rhythm in a Computer-Assisted Composition System," *Computer Music Journal*, vol. 31, no. 4, pp. 59–70, Dec. 2007. [Online]. Available: http://www.mitpressjournals.org/doi/abs/10.1162/comj.2007.31.4.59

[26] B. Degazio, "A Computer-based Editor for Lerdahl and Jackendoff's Rhythmic Structures," *Proceedings of International Computer Music Conference*, 1996.

[27] N. Collins, "A Microtonal Tempo Canon Generator After Nancarrow and Jaffe," in *Proceedings of the International Computer Music Conference*, 2003.

[28] I. Xenakis, "More Thorough Stochastic Music," *Proceedings of International Computer Music Conference*, 1991.

[29] D. P. Creasey, D. M. Howard, and A. M. Tyrrell, "The Timbral Object - An Alternative Route to the Control of Timbre Space," *Proceedings of International Computer Music Conference*, 1996.

[30] N. Osaka, "Toward Construction of a Timbre Theory for Music Composition Composition," *Proceedings of International Computer Music Conference*, 2004.

[31] J. C. Seymour, "Computer-assisted Composition in Equal Tunings: Tonal Congnition and the *Thirteen Tone March*," *Proceedings of International Computer Music Conference*, 2007.

[32] A. Gräf, "On Musical Scale Rationalization," *Proceedings of International Computer Music Conference*, 2006.

[33] C. Ariza, "Ornament as Data Structure : An Algorithmic Model Based on Micro-Rhythms of Csángó Laments and Funeral Music Music of the Csángó," *Proceedings of International Computer Music Conference*, 2003.

[34] W. Chico-Töpfer, "AVA: An Experimental, Grammar/Case-based Composition System to Variate Music Automatically Through the Generation of Scheme Series," *Proceedings of International Computer Music Conference*, 1998.

[35] N. Collins, "Musical Form and Algorithmic Composition," *Contemporary Music Review*, vol. 28, no. 1, pp. 103–114, Feb. 2009.

[36] L. Smith, "SCORE- A Musician's Approach to Computer Music," *Journal of the Audio Engineering Society*, vol. 20, no. 1, pp. 7–14, 1972.

[37] H.-W. Nienhuys and J. Nieuwenhuizen, "LilyPond, A System for Automated Music Engraving," in *Proceedings of the XIV Colloquium on Musical Informatics (XIV CIM 2003)*. Citeseer, 2003, pp. 167–172.

[38] H. H. Hoos, K. Hamel, K. Renz, and J. Kilian, "The GUIDO Notation Format- A Novel Approach for Adequatly Representing Score-level Music," *Proceedings of International Computer Music Conference*, 1998.

[39] K. Hamel, "NoteAbility Reference Manual," 1997.

[40] M. Kuuskankare and M. Laurson, "Expressive Notation Package," *Computer Music Journal*, vol. 30, no. 4, pp. pp. 67–79, 2006. [Online]. Available: http://www.jstor.org/stable/4617984

[41] AVID. Plugins for Sibelius. [Online]. Available: http://www.sibelius.com/download/plugins/index.html?help=write

[42] E. Selfridge-Field, *Beyond MIDI: The Handbook of Musical Codes*. The MIT Press, 1997.

[43] N. Consortium and et al., "NIFF 6a: Notation Interchange File Format," NIFF Consortium, Tech. Rep., 1995.

[44] M. Good, "MusicXML for Notation and Analysis," in *The Virtual Score: Representation, Retrieval, Restoration*, ser. Computing in Musicology, W. B. Hewlett and E. Selfridge-Field, Eds. MIT Press, 2001, no. 12, pp. 113–124.

[45] D. Cantor, "A Computer Program that Accepts Common Musical Notation," *Computers and the Humanities*, vol. 6, no. 2, pp. 103–109, 1971.

[46] L. Polansky, "HMSL (Hierarchical Music Specification Language): A Theoretical Overview," *Perspectives of New Music*, vol. 28, no. 2, 1990.

[47] G. Krasner, "Machine Tongues VIII: The Design of a Smalltalk Music System," in *The Well-tempered Object: Musical Applications of Object-oriented Software Technology*, S. T. Pope, Ed. MIT Press, 1991.

[48] A. Mahling, "How to Feed Musical Gestures into Compositions," *Proceedings of International Computer Music Conference*, 1991.

[49] G. Diener, "Addendum: A Hierarchical Approach to Music Notation," in *The Well-tempered Object: Musical Applications of Object-oriented Software Technology*, S. T. Pope, Ed. MIT Press, 1991.

[50] ——, "TTrees: A Tool for the Compositional Environment," in *The Well-tempered Object: Musical Applications of Object-oriented Software Technology*, S. T. Pope, Ed. MIT Press, 1991.

[51] ——, "Nutation: Structural Organization Versus Graphical Generality in a Common Music Notation Program," *Proceedings of International Computer Music Conference*, 1989.