# Notating Stochastic Music with DISSCO

*Andrew Orals and Dr. Sever Tipei*

Computer Music Project of the School of Music
National Center for Supercomputing Applications
University of Illinois

ABSTRACT

The Digital Instrument for Sound Synthesis and Composition (DISSCO) unifies algorithmic composition with score and sound file output in one seamless process. Within DISSCO, the composition module employs a tree structure to represent the piece in layers of abstraction. An important problem is converting the internal structure of the piece into Western musical notation. This has been a topic of considerable research, devising methods for reliably producing notation output. This paper details a new design which extends these notation methods to multiple distinct sections and builds a foundation for future contribution and extension.

## 1. INTRODUCTION

### 1.1 DISSCO

The Digital Instrument for Sound Synthesis and Composition, abbreviated DISSCO, is software for computer-assisted algorithmic composition. DISSCO unifies the process of composition by cooperating three modules, namely the composition module (CMOD), the library for additive synthesis (LASS), and the graphical user interface (LASSIE). DISSCO also offers a comprehensive approach by taking composition parameters from the user, performing computations, and then outputting a finished product as a seamless process [4].

While DISSCO allows music of any style with both stochastic and deterministic parameters, it exhibits a strong bias towards pre-planned compositions employing controlled randomness. Such compositions constitute "Manifold Compositions," wherein one set of parameters can produce a variety of interpretations [3]. These variations correspond to unique random seeds. DISSCO's unified and comprehensive approach reflects its underlying "black box" design principle: after the user designs the parameters of the composition, the user may not interrupt or otherwise participate in the process. Doing so would falsify the experiment and subvert the conceptualization of the piece as a manifold composition [4]. In this sense, DISSCO acts as a companion to the composer, complementing the skill of the user in creating parameters with the computer's aptitude in pseudo-randomness and speed [2].

### 1.2 Elementary Displacement Units and Sieves

Numerical sieves are a powerful and flexible tool in discrete mathematics to filter sets that can be mapped to the natural numbers based on numerical equivalence classes. To place objects in metric time, DISSCO employs a grid of Elementary Displacement Units (EDU's) and filters values from this grid using sieves, as shown by Xenakis [6].

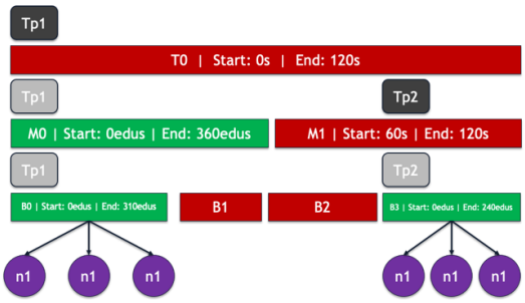
The Elementary Displacement Unit is the basic unit of rhythm in an exact event. By defining the number of EDU's per beat and the note representing the beat through a time signature, a rhythmic system can be created. Then, sieves can select certain rhythms and discard others. For example, consider a time signature where each quarter note represents 6 EDU's. A sieve of $6_0$ allows only quarter notes, where 6 is the "modulus of symmetry" and the index 0 represents a shift from the reference point. A sieve of $6_3$ then allows only eighth notes. Using Boolean operations, sieves can be combined to construct more discerning rhythmic systems. For instance, a sieve of the form:

$$6_0 \cup 6_2 \cup 6_3$$

would allow quarter notes, eighth notes, and eighth note triplets.

## 1.3 DISSCO's Event Structure

Internally, CMOD employs a tree directed graph structure. Each node in the tree is an *Event* object, while the leaves are *Bottom* objects, a derivative of the *Event* class. These *Events* represent the composition as musical concepts of varying abstraction, composed of a name, start time, tempo, duration, and a collection of child events. All *Events* hold a global start time relative to the start of the piece and a duration in seconds, designated 'inexact,' and some *Events* also hold an integer start time and duration in EDU's, designated 'exact.' *Events* are categorized generally as Top, High, Mid, Low, and Bottom, and the *Events* can represent any layer of abstraction that the composer wishes so long as the Top event represents the entire piece and there is at least one *Bottom* event as a leaf node in the tree [5]. *Bottom* events cannot be internal nodes. For example, the children of the Top event might represent movements of a piece, the next level of abstraction might represent sections of these movements (i.e. the development section of sonata form), and a *Bottom* event might represent a specific melody in the composition. This makes the form of the composition a natural product of the data structure.




In addition to *Events*, CMOD defines a *Tempo* object which has its own global start time in seconds, defining the number of beats per bar, the note type which represents a beat, and the number of EDU's per beat. A *Tempo* can have a start time anywhere between the beginning and ending of the piece in seconds. Since an *Event's* exact start time in EDU's is relative, the *Tempo* object acts as an anchor point defining the beginning of the exact *Event*. Without a corresponding *Tempo*, the EDU's within an exact *Event* are meaningless.

The possibility of nested exact *Events* within inexact events presents some special cases. Whenever there is an exact child with an inexact parent, this triggers a new *Tempo* object to be created in the parent as a reference point for the child. Furthermore, whenever there is an exact child with an exact parent, the child inherits the parent's tempo to prevent implicitly nested tempos, as the *Events* are related by nature of them both being exact.

## 1.4 The Problem

Although DISSCO's manifold compositions lend themselves naturally to sound synthesis, notating output using western notation, while preserving the underlying intention of the musical events, has been a topic of considerable research yielding substantial progress. Previous work has developed reliable methods for notating single musical lines of uniform tempo and time signature; however, this work assumes that the *Bottom* events of the entire piece are exact [2]. Furthermore, because the implementation was developed by many contributors at different times, the notation logic was very tightly coupled with code for the *Bottom* and *Note* classes, making it difficult to extend and modularize. Therefore, the goal of this research is to add functionality to notate distinct exact sections of the piece and improve the extensibility and documentation of the notation module. To this end, the following problems must be considered:

1. Exact sections can be separated by inexact time spans, necessitating the ability to instantiate notation logic for each section.

2. Notes added to each section may not be inserted into the notation score in any order while they must be notated in chronological order.
3. Between exact sections, there is likely a gap which is not necessarily expressible in the previous section's rhythmic system.

## 2. NOTATION MODULE DESIGN

### 2.1 The ***Section*** Class

To allow notation logic to be instantiable for distinct exact sections, all the notation functionality previously implemented as static functions was refactored and decomposed into private member functions of a *Section* class. In the public interface, a clear distinction is made between the processes of inserting notes and building the score. Internally, a simple Boolean variable indicates whether the section is built, allowing notes to be inserted in any order while protecting against corrupting the section after building. This not only makes the code more intuitive for future contributors, but also makes the code easier to diagnose by reducing the possibility of side effects, or functions modifying data outside their scope.

### 2.2 The ***NotationScore*** Class

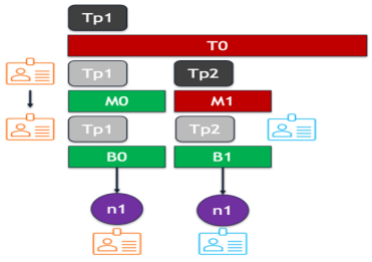
The *NotationScore* class manages *Section* classes by facilitating adding new sections to the score, inserting notes into the score, building the score, and outputting the final product to a text file. All these operations are clearly delimited by their own respective functions in the public interface.

Since the root exact *Event's Tempo* object is inherited throughout exact sections of CMOD's tree structure, distinct tempos represent different exact sections of the score separated by some inexact time span. Since all *Tempo* objects have a global inexact start time in seconds and every exact *Event* in the tree holds its own *Tempo*, every *Event* can attempt to add its own *Tempo* to the score. Then, the job of the *NotationScore* is to only construct and add sections from unique *Tempos* and keep the *Section* objects sorted in chronological order using their inexact global start time.

## 3. SOLUTIONS TO PROBLEMS IN SCORE BUILDING

### 3.1 Distributing *Note* Objects into Sections

After all Sections are added to the score, an important problem is inserting *Notes* into the sections. Like the addition of new sections to the score, *Notes* are not necessarily added to the score in chronological order. Furthermore, while the *Notes* are constructed in the *Bottom* class consistently after their corresponding *Tempo*, it would constrain future development to rely on this obscure artifact of the class design to ensure that *Notes* are added to the correct section. One solution to the problem is to use the notes' global inexact start time to distribute them into sections with a corresponding time span. However, this could prove especially limiting in the future if layering is implemented in the *NotationScore*. Instead, the most reliable solution is to assign a unique identifier to both the *Note* and its corresponding *Tempo*. This identifier is essentially a pointer to the root exact ancestor *Event* of the section. For example, in the diagram below, the root exact ancestors would be M0 and B1. The Notes' identifiers can then be compared against those of the sections to determine into which the note can be inserted.



### 3.2. Joining Notation Sections

When two consecutive notation sections occur, suppose S1 and S2, S2's start time is determined in EDU's with respect to S1's time signature and tempo. For example, if S1 has 6 EDU's per beat at 60 beats per minute and S2 starts at 4 seconds globally, S2 then starts at 24 EDU's and S1 is given an allotment of 24 EDU's to fill during the build stage. Once S1 is built, S1 will have used some quantity of its allotted EDU's leaving a remainder $r$, and the last bar with some quantity of EDU's $b$. If $r < 0$ then the sections overlap and an error is produced, while if both $r$=0 and $b$=0, the sections align perfectly. In all other cases, a bar of shortened or lengthened time must be placed between the sections to accommodate the excess time requiring the following steps be taken:

1. Iterate through each possible power of 2 that evenly divides the EDU's per beat of S1; let the power of 2 be $p$ and the result of the division be $d$.
2. If a multiple of $d$ equals the total EDU's to use ($r$+$b$), create a new time signature with a numerator of $(r + b) / d$ and a denominator of the unit note of the previous section multiplied by $p$; return.
3. If a multiple of $d$ does not divide the total EDU's, find the multiple of $d$ that would encompass the total EDU's and record the error as $\epsilon$.
4. Create a new time signature from the numerator and denominator that gave the smallest value of $\epsilon$.

The last bar is deleted from the flattened section and its constituent notes are added to a new, nested section made from the previously calculated time signature. The nested section is then built, and its contents are appended to those of the enclosing section. In the case that the calculated time signature is the same as the previous time signature (i.e. the sum of the remaining time and the time span of the last bar is a full bar) the time signature need not be notated.

## 4. CONCLUSION AND FURTHER WORK

This research has significantly increased the compatibility between the notation module and the full suite of features available in DISSCO, laying the foundation for extensions in the future. Using the class design outlined in this paper, it should be straightforward to implement aesthetic improvements in the output score, such as tempos and articulations. A logical extension is implementing a metadata class storing information about the score. Finally, further research could implement multiple concurrent lines of notation in the score.

## 5. REFERENCES


1. Sun, Haorong, & Tipei, Sever. “Automatic Notation of Complex Rhythms Using Sieves in DISSCO.” 2019 WOCMAT Conference, WOCMAT, 1 Dec. 2019, lewis841214.github.io/WOCMAT2019.github.io/.
2. Tipei, S., The Computer, a Composer's Collaborator, Leonardo Journal of the International Society for the Arts, Sciences, and Technology, vol.22, no. 2, pp. 189-195, 1989
3. Tipei, Sever. “XXII Generative Art 2019 Conference.” Domus Argenia Publisher, ISBN 978-88-96610-39-8, Manifold Compositions and the Evolving Entity, 2019, pp. 369–375.
4. Tipei, Sever. “Conceiving Music Today: Manifold Compositions, DISSCO and Beyond.” 11th WSEAS International Conference on Acoustics & Music: Theory & Applications (AMTA '10), "G. Enescu" University, Iasi, Romania, June 2010. WSEAS Mechanical Engineering Series.
5. Tipei, Sever. “Composition as an Evolving Entity; an Experiment in Progress.” Proc. 2016 ICMC Int'l Computer Music Conference (Utrecht, The Netherlands, September 2016).
6. Xenakis, I. - Formalized music, Pendragon Press, 1992. p. 268.