

Mondrian Music Description Language And Sequencer

Peter Brinkmann
Department of Mathematics
The City College of the City University of New York
New York, NY 10031
U.S.A.
brinkman@sci.ccnycuny.edu

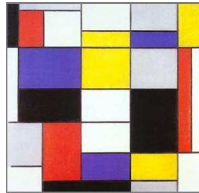


Figure 1: Piet Mondrian, Composition A, 1923

Abstract

The Mondrian Project implements musical instruments on top of computer text editors. It consists of a music description language and interactive MIDI sequencers that may be used as editor plugins for both vim and Emacs.

1 Introduction

Most MIDI sequencers and music description languages organize music as superimposed layers, mimicking traditional notation. A more flexible approach would allow users to create music by lining up a few notes sequentially, stacking such lines on top of each other, aligning such stacks sequentially, stacking the resulting compound objects on top of each other, etc. In other words, it would structure music like one of Piet Mondrian's paintings (Figure 1).

The Mondrian Project supports the expression of such structures, and it provides a set of tools for recording and performing. It consists of a language interpreter, the *frontend*, and various output devices, the *backends*.

Available backends include MondrianLive and MondrianPM, two interactive sequencers that act as an arpeggiator, pattern sequencer, etc. MondrianLive uses the sequencer API of the Advanced Linux Sound Architecture (www.alsa-project.org), while MondrianPM uses portmidi (www.cs.cmu.edu/~music/portmusic). Mondrian-

Live is more powerful but limited to Linux; MondrianPM currently lacks a few features but should work across all popular platforms.

Both MondrianLive and MondrianPM can be used as editor plugins for both vim (vi improved, www.vim.org) and Emacs (www.gnu.org/software/emacs).

2 Related work

There are a number of music description languages (e.g., ABC (Walshaw, Chris 2005), CSound's orchestra file format (Boulanger 2000), etc.), but none of them seems to offer support for Mondrian-like structures, relative steps through user-defined scales, or reusability of small chunks of code. LilyPond (Nienhuys and Nieuwenhuizen 2003) offers some of the desired structures, but it is geared toward typesetting. The syntax of Mondrian bears a superficial resemblance to David Griffiths' macro language (Alexander et al. 2004).

Mondrian applies some of the ideas behind Donald E. Knuth's \TeX typesetting system (Knuth 1984) to a musical setting. \TeX formats text by placing letters in boxes whose contents may move either vertically or horizontally (vbox and hbox) and fills a page by lining up vboxes in hboxes, stacking up hboxes in vboxes, etc.

3 Language

The Mondrian interpreter has a state comprising several properties, many of which are familiar MIDI values. User-accessible properties include the current note, (off) velocity, track, and MIDI channel. Mondrian's state also includes a scale (major, minor, pentatonic, etc.) that users can define arbitrarily.

Mondrian's syntax is inspired by vim commands. Generally speaking, Mondrian commands are words or characters

Command	Action
>	move up n steps in scale
<	move down n steps in scale
+	move up n half steps
-	move down n half steps
c	set channel to n
n	set note to n
T	set tempo to n bpm (1..600)
v	set velocity to n

Table 1: Common commands; commands are of the form nx , e.g., 3c, 120T

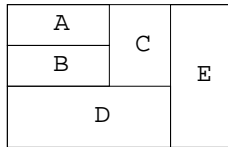


Figure 2: Example: `[[[A | B] C] | D] E`

that may be preceded by an integer n , such as 160T, or 3+. Table 1 lists some common commands.

Note events are triggered by numbers that are not followed by a state-changing command, i.e., a note event is denoted by an integer followed by space, tab, newline, or a few other special characters.

A number n defines the duration of a note in the following way: If n consists of just one digit, then the duration is 2^{-n} , e.g., 0 means a whole note (of the current pitch, velocity, etc.), 1 a half note, 2 a quarter note, etc. If n consists of more than one digit, then the last digit serves as an exponent, and the rest as a multiplier. For instance, 54 means $5 \cdot 2^{-4} = \frac{5}{16}$, 33 means $3 \cdot 2^{-3} = \frac{3}{8}$, 172 means $17 \cdot 2^{-2} = \frac{17}{4}$, etc.

If a sequence of commands is enclosed in parentheses (), then any state changes inside the parentheses are local to the contents of the parentheses, i.e., after the closing parenthesis the state reverts to what it was at the opening parenthesis.

3.1 Lines and stacks

When Mondrian encounters a note event, it schedules the event at the current time and advances the time by the duration of the note, so that note events are played consecutively. Brackets [] and vertical bars | are used to stack such lines on top of each other and play them simultaneously. When Mondrian encounters an opening bracket, it saves the current time t_0 . When it encounters a vertical bar, it resets the time to t_0 . When it encounters a closing bracket, it advances the time to t_0 plus the duration of the longest line enclosed by the brackets.

This construction can be repeated indefinitely, i.e., one can stack up lines, align stacks and notes in sequences, then form stacks of such compound sequences, yielding structures like Mondrian’s paintings (Figure 2).

3.2 Macros

Macros are defined by enclosing the contents of the macro in exclamation marks, followed by a label. For instance, `!127v!fff` will define a new macro called `fff`, and calling it will set the velocity to 127 (fortissimo). If a number n precedes the label of the macro, then the macro is called n times.

3.3 Example

Here are the first four notes of a popular canon, *Frère Jacques* (Figure 3): `2 > 2 > 2 2< 2`

The initial state of Mondrian has current note C and current scale C major. The first character, 2, triggers a quarter note of the current pitch. The greater sign > moves the current note up in the scale, i.e., the note is now D. The next number, 2, triggers a quarter note D, and the next two characters trigger a quarter note E. The next two characters, 2<, take us two steps down on the scale, back to C, and the final character triggers a quarter note C, so that we’ve defined the sequence CDEC.

Here’s the beginning of the second line (“dormez-vous”): `(2> 2 > 2 > 1)`

We first move up two steps, so that the note is E, play a quarter note E, then a quarter note F, and finally a half note G (because $2^{-1} = \frac{1}{2}$). The opening parenthesis saves the current state, and the closing parenthesis restores it, so that we’re back to note C even though the line ended in G.

The next line begins like this (“sonnez les matines”): `(4> 3 > 3 < 3 < 3 < 2 2< 2)`

There are four eighth notes in this line (because $2^{-3} = \frac{1}{8}$).

Finally, here is the beginning of the last line (“ding ding dong”): `(2 3< 2 3> 1)`

Since there is a lot of repetition in the song, we define some macros that represent the basic building blocks:

```
!(2 > 2 > 2 2< 2)!fj
!(2> 2 > 2 > 1)!dv
!(4> 3 > 3 < 3 < 3 < 2 2< 2)!slm
!(2 3< 2 3> 1)!ddd
```

The contents of macros are enclosed in exclamation marks, and the label of the macro follows the second exclamation mark.

Now we express the melody in terms of these macros: `fj fj dv dv slm slm ddd ddd`

2 >2 >2 2<2
Frè - re Jac-ques

2>2 >2 >1
Dor - mez vous

4>3 >3 <3 <3 <2 2<2
Son-nez les ma-ti - nes

2 3<2 3>1
Ding ding dong

Figure 3: The first measure of *Frère Jacques*, with four superimposed voices. Note that the code in the figure contains fewer spaces than the code in the text; spaces may be omitted unless doing so would create ambiguities.

We can save a few keystrokes by calling each macro twice: `2fj 2dv 2slm 2ddd`

We now introduce more voices to our canon:

```
2fj
[2dv | 2fj]
[2slm | 2dv | 2fj]
[2ddd | 2slm | 2dv | 2fj]
```

The first voice starts all on its own. After the first two measures (`2fj`) the second voice comes in. The opening bracket `[` remembers the current time t_0 . We enter the third and fourth measure of the first voice (`2dv`). The vertical bar `|` takes us back to time t_0 , such that the first two bars of the second voice (`2fj`) are stacked on top of the third and fourth bars of the first voice. The closing bracket `]` indicates the end of this stack. The time is now the end of the fourth measure of the first voice. The remaining lines introduce more and more voices. Figure 3 shows the first measure defined by the last line.

This example illustrates the basic idea in a nutshell: We have macros that contain sequences of notes. Then we stack such sequences on top of each other, and finally we align the stacks after one another.

4 Backends

Mondrian comes with three different backends. The simplest one is the MIDI file writer. Called from the command line, it reads Mondrian code from a file, say `in.mon`, and writes the result to a MIDI file:

```
python mondrian.py -o out.mid in.mon
```

If no filenames are given, the MIDI file writer will read from `stdin` and write to `stdout`, so that it may serve as a filter in pipes.

The Mondrian website (www.sci.ccny.cuny.edu/~brinkman/software/mondrian/) includes an online demo that calls the MIDI file backend from a CGI script. Users can enter Mondrian code or upload a file and receive a MIDI file in return.

4.1 Interactive MIDI sequencers

There are two interactive sequencer backends, MondrianLive and MondrianPM. For the purposes of this note we will focus on MondrianLive.

MondrianLive keeps a queue of chunks of music (a chunk might be anything; a note, a measure, an entire song, etc.) that are played in sequence. Whenever the user adds a new chunk, it is appended to this queue and will be played after all previous chunks have been played. When MondrianLive reaches the end of this queue, it keeps repeating the last chunk, unless the last chunk is empty, in which case playback will be paused until the next nonempty chunk arrives.

In addition to plain Mondrian code, MondrianLive understands a number of commands of the form

```
## command parameters
```

Since the Mondrian interpreter ignores comments starting with `#`, MondrianLive commands do not affect the interpreter.

The following is a partial list of available commands:

map maps tracks to MIDI devices. For example, the command

```
## map timidity 0 400 2
```

will map track 2 to input port 0 of TiMidity++ (assuming that TiMidity++ is already running in server mode), with a latency correction of 400ms.

imap connects the input port of MondrianLive to a MIDI input device, typically a keyboard.

record records MIDI events that arrive at MondrianLive's input port and converts them to Mondrian code.

4.2 Editor plugins

MondrianLive can function on its own, taking its input from a named pipe, but it is primarily intended to be used as a plugin for vim or Emacs. We will only discuss the vim plugin here; the Emacs plugin is similar.

If the vim plugin is installed, then vim will load it and launch the MondrianLive sequencer upon opening a file with a `.mon` extension.

Pressing a function key (`F4`) will cause vim to send the current line into MondrianLive, which will interpret it and append it to its queue of chunks of music. If the user marks a range of lines (e.g., using vim's visual mode) and presses `F4`, then the entire range will be sent to MondrianLive.

5 Implementation

Mondrian is written in Python (van Rossum and Drake 2003), with C extensions handling the parts that are critical to timing. It requires the following packages:

PySeq Python bindings for the ALSA sequencer API (Brinkmann 2005), MondrianLive backend only, www.sci.ccny.cuny.edu/~brinkman/software

pyPortMidi Python bindings for PortMidi (MondrianPM backend only), www.python.org/pypi/pyPortMidi

Pymacs Python as an extension language for Emacs, Emacs sequencer plugin only, pymacs.progiciels-bpi.ca

Python Midi Package Python library for parsing MIDI files, MIDI file backend only, www.mxm.dk/products/public/pythonmidi

Mondrian is freely available under the GNU General Public License, at www.sci.ccny.cuny.edu/~brinkman/software.

6 Outlook

There are three major additions to Mondrian that I would like to implement in the future.

MIDI synchronization Since text-based approaches are not suitable for handling binary data such as audio signals, it would be desirable to synchronize Mondrian with audio workstations like Ardour (www.ardour.org).

Stochastic filters Mondrian currently lacks a good mechanism for ameliorating the mechanical feel of MIDI

files. Hooks for stochastic filters (Lorrain 1979) would be a valuable addition.

Improved macro extraction Mondrian comes with a tentative MIDI-to-Mondrian translator that extracts recurring structures as macros. A better implementation would be faster, and it might provide a way of mining MIDI files for reusable elements such as drum patterns or bass lines.

One might create and share libraries of small reusable chunks of Mondrian code. Since notes can be defined in a relative fashion, by stepping up or down through user-defined scales, such building blocks are easy to transpose and adapt for reuse. Since meaningful chunks of code are small enough to be pasted into instant messages, one might combine a text editor like vim, MondrianLive, and an instant messenger client like gaim to have distributed jam sessions.

7 Acknowledgments

I would like to express my gratitude to Camille Goudeseune for patiently serving as a sounding board throughout the development process, and to the members of the LAU and LAD mailing lists for all their help and support. Figure 3 was created with LilyPond (www.lilypond.org).

References

- Alexander, A., N. Collins, D. Griffiths, A. McLean, F. Olofsson, J. Rohrhuber, and A. Ward (2004). Live algorithm programming and a temporary organisation for its promotion. In *Read Me Software Art and Cultures Conference*. Aarhus, Denmark.
- Boulanger, R. E. (2000). *The Csound Book: Perspectives in Software Synthesis, Sound Design, Signal Processing, and Programming*. The MIT Press.
- Brinkmann, P. (2005). MidiKinesis — MIDI controllers for (almost) any purpose. In *LAC2005 Proceedings*, pp. 9–12. Karlsruhe, Germany: Zentrum für Kunst und Medientechnologie.
- Knuth, D. E. (1984). *The TeXbook*. Addison-Wesley Professional.
- Lorrain, D. (1979). A panoply of stochastic cannons. *Computer Music Journal* 3.
- Nienhuys, H.-W. and J. Nieuwenhuizen (2003). LilyPond, a system for automated music engraving. In *Proceedings of the XIV Colloquium on Musical Informatics (XIV CIM 2003)*. Firenze, Italy.
- van Rossum, G. and F. L. Drake (2003). *The Python Language Reference Manual*. Network Theory Ltd.
- Walshaw, Chris (2005). the abc musical notation language. <http://staffweb.cms.gre.ac.uk/~c.walshaw/abc/>.