Vol.2019-MUS-123 No.17
Vol.2019-SLP-127 No.17
2019/6/22

音楽プログラミング言語のソースコードを楽譜と捉え、それを編集するツールの構想

松浦 知也[1,a)]　城 一裕[2,b)]

概要：本研究では音楽プログラミング環境のソースコードを音楽の構造を表す記述≒楽譜として捉え、それを時間軸に沿って編集するツール、そのための専用プログラミング言語と実行環境を提案する。2000 年前後を機としたコンピューターの性能の向上に伴い、1) 音価からサンプルまでの広い時間スケールを統合して扱える、2) 高度な音響処理を低レベルの言語に依存せずに記述できる、3) コードをリアルタイムで書き換えて音響生成へと即時に反映できる、音楽プログラミング環境が登場してきた。現在では、3) のライブコーディング機能を用いてダンスミュージックを生成する音楽イベント Algorave に見られるよう、これまでの実験的な音楽だけでなく、ポピュラー音楽においても音楽プログラミング環境が使われるようになってきている。本稿ではこれらの歴史を踏まえ、音楽プログラミング言語の課題として音楽の構造を分かりやすく、かつボトムアップ的に定義できる言語の必要性について議論し、その言語仕様の案を提示する。

Tomoya MATSUURA[1,a)]　Kazuhiro JO[2,b)]

1. はじめに

今日、音楽を制作するプロセスの中でコンピューターを用いることは日常的になっている。楽譜を制作することから音色の微妙な質感をコントロールするための信号処理まで、音楽を形作る多くの要素がコンピューターのプログラムによって実現されている。こうしたコンピューター上で音を生成する手段の中で、音楽に特化したプログラミング言語を使用して音を生成するという手法がある。

このための環境で代表的なものとしては Max[*1]、Puredata[*2]、SuperCollider[*3]などが挙げられ、2000 年以後は ChucK[1] のように音価からサンプルまでの広い時間スケールを扱えるもの、Faust のように高度な音響処理を C などの低レベルの言語に依存せずに記述できるもの [2]、また ChucK や Extempore[3] のようにコードをリアルタイムで書き換えて音響生成へと即時に反映できる言語が登場している。Dannenberg はこうした近年の音楽プログラミング言語の発展を概観しこれからの課題として、

1. MIDI ノートのような時間方向に離散的に発生するイベント処理と、音声信号のように時間方向に一定間隔で発生する処理とを統合して記述できる言語
2. モジュールが内部変数 (ステート) を保持するようなプログラミングモデルの方が直感的には自然だが、内部変数を持たず全てのパラメータが入出力で表される関数型のモデルの方がデータ処理としては自然かつ効率的という対立構造
3. リアルタイムに変化する複雑な音楽プログラムの内部状態を直感的にわかりやすく観測できる機構を持つこと

を挙げている [4]。

これらの課題は Dannenberg が同論文中で音楽のプログラミングは一般的なプログラミングのように可能な限り速く計算を完了させれば良いのではなく、ある時間に答えを出さなければいけないという違いがあると述べるように、そもそも音楽という時間軸に沿って変化するものを如何にソースコードという不変なものに抽象化するかという根本的な課題がある為だと考えられる。言い換えると、**音楽プログラミング言語のソースコードが楽譜のように対象の音楽の構造をわかりやすく表せる機能を持つ事**が求められる。

かつ、音楽を作るための道具として、音楽プログラミン

[1] 九州大学大学院芸術工学府
Graduation School of Design, Kyushu University
[2] 九州大学芸術工学研究院
Faculty of Design, Kyushu University
[a)] me@matsuuratomoya.com
[b)] jo@jp.org

[*1] https://cycling74.com/products/max-features
[*2] http://puredata.info
[*3] https://supercollider.github.io/

Vol.2019-MUS-123 No.17
Vol.2019-SLP-127 No.17
2019/6/22

グ言語は常に新しい音楽表現の可能性を担保できるよう、**基礎となるモジュールが既存の音楽表現を元にして作られるトップダウン的な構造では無く、その言語の中でボトムアップ的に定義できるようにする**べきである。Max や SuperCollider はこの点において、例えば、フィルターやオシレーターなどの基本的なオブジェクトが C++などより低レベルの言語で記述されており、Max 上でそれ自体を編集するといったことは出来ない。信号処理の効率を考えると低レベル言語で記述することは有利ではあるが、Faust や Kronos[5] が LLVM[6] バックエンドを用いることで、リアルタイムで高度な信号処理をコンパイルして実行できているように、パフォーマンスを理由に基本モジュールを別言語で定義する意義は今日では比較的小さくなっていると言える。

こうした、音楽の構造を分かりやすく、かつボトムアップ的に定義し記述する言語の先行例としては、音楽プログラミング言語中の基本的な型として時間を扱うというアプローチがある。Chronic[7] は関数型プログラミング言語 OCaml 上で音楽を扱うためのライブラリで、`event`,`vec`,`ivec` という 3 種類の時間構造を表すコンストラクタを持ち、さらにそれらを組み合わせた型を定義していくことで複雑な時間構造を表現することができる。しかし時間軸上に不均一に存在するイベントの集合などを表すことは難しく、またそのデータ構造を処理する手続きも、関数型言語の慣例に沿ったものとなっている為に、対象ユーザーとして音楽家のようなプログラミングそのものを専門としていない人を想定した時には難易度が高い。

このような状況を踏まえ筆者は、音楽の構造を表す事に重点を置いた新しい音楽プログラミング環境の提案を行う。具体的には、Dannenberg の課題 1. 2. の解決策として全ての変数に対して時間軸上の位置情報を付随させることで、時間方向に離散的なイベントと連続的なイベントの処理を統合して記述でき、かつソースコードの中のデータ構造と処理の分離をより明確にする音楽プログラミング言語を提案する。また、課題 3. に対する解決策として、GUI でそのプログラミング言語のソースコードの構造を視覚的に表し、かつテキストと双方向に編集ができるエディターソフトウェアと組み合わせるという環境を提案する。

以下本稿では、提案するツールの全体の構造及びそのプログラミング言語の中でも根幹を成す、変数に対し時間を付随させる機能の案を中心に議論する。

2. 提案するツールの概要

提案するツールの概要は図 1 のようなものである。ソースコードはコンパイラーで評価され、オーディオや MIDI などの信号を出力する。エディターソフトウェアはソースコードを読み込み、その構造を表示、また GUI を用いて編集することができる。この仕組みには Chugh らの Sketch-n-Sketch というベクター画像をプログラムとダイレクト・マニピュレーション双方向から操作するソフトウェアで用いられている Bidirectional Programming[8], [9] という手法を参考に開発する予定である。現在構想中のインターフェースは Digital Audio Workstation(DAW) ソフトウェアのように左から右に時間軸が表示されるようなものをベースにしているが、ソースコードの編集ツールという形をとる事で、将来的に同じ音楽作品でも全く違うインターフェースを持つエディターを用いて再編集、リミックスができるといった可能性を担保することができる。

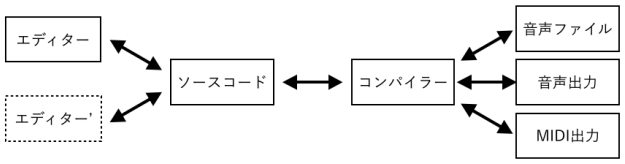

図 1　提案するツールの全体の構成を表した図。

2.1 変数に対する時間の付随

提案する言語の中では全ての変数に対して`@`の後ろに時間の絶対位置を付随させることができる。時間情報には `[]` を用いて時間の単位を指定できる。時間は配列や、適用された関数に対しても指定できる (図 2)。

```
somevar = 1@2.5[second]
somearray = [5,10,23,2]@1[second]
somenumber = 2
someapp = somefun(somevar)@2[second]
```

また時間の付随はネストさせることもできる。

```
somevar_multitime =1@2.5[second]@5.5[second]
somevar_multitime =1@[2.5,5.5][second] //省略形
```

さらに、時間の付随は配列の初期化式でも定義できる。これは Haskell の無限リスト[*4]の記法を参考にしており、遅延評価されることで無限に続く形をとることができる。オーディオ波形のような一定間隔で並ぶデータはこれを用いて定義することができる。

```
somevar_sequence = 24@[1..]
somefile = readwav("sample.wav")@[1..][sample]
```

2.2 関数適用と時間の消費

時間を付随させた変数を関数に適用させると、その適用した関数に時間を付随したことと同じになる。

```
somevar = 1@2.5[second]
incl(x) = x+1
// incl(1)@2.5と同じこと
incl(somevariable)
```

[*4] https://en.wikibooks.org/wiki/Haskell/Lists_II#Infinite_Lists

@が付いた変数は call の引数として評価されると、そのタイミングに実行されるのではなくスケジューラーに実行するタイミングが登録される。

例えば、MIDI ノートのように始まりの時間と終わりの時間、音程という 3 つのパラメーターを持つデータは次のように定義できる (単純化のため、音の強さについては省略する)。

```
mynote = 78@[1.2,3.6]
```

このデータを処理する関数として noteOn と noteOff という関数定義を考える。

```
//時間1.2の時に実行
noteOn(note,instrument) = {
  set(instrument.gain,1)
  set(instrument.pitch,note)
  //noteは関数の中では1.2が消費され、78@3.6として
    扱われる
  call(noteOff(note))
}
noteOff(note,instrument) = {
  set(instrument.gain,0)
}
call(noteOn(mynote,myinstrument))
```

この時、noteOn 関数は時間 1.2 の時に実行され、その関数内では mynote は引数 note として使われるときに@1.2 が取れて 78@3.6 となり、新たに noteOff 関数に渡される。

この一連の流れは Extempore の Temporal Recursion([3], p83) というコンセプトを基にしている。Temporal Recursion は関数内で (callback 時間 関数) を呼び出すと指定した時間後に関数を呼び出すようスケジューラーに登録するもので、定義する関数自身を再帰的に呼び出すことで、例えば一定間隔で無限に発音するような操作を記述できる。

Temporal Recursion は callback という関数に次に呼び出す時間が結びついている。一方で提案する手法では call 関数に渡すのは noteOff という次の操作だけであり、いつ実行されるかは元々の引数自体に付随した時間によって決まる。この記法により音楽の構造の記述における**手続きとデータの分離を明確にする**ことができる可能性を持つ。

3. おわりに

本稿では音楽プログラミング言語の課題として音楽の構造を分かりやすく、かつボトムアップ的に定義できる言語の必要性を述べた上で、ソースコードを楽譜のように扱い、それを GUI を用いて操作するツールの構想を述べた。加えて、その根幹となる変数に対して時間を付随させる方式を提案した。本ツールは未だ実装の初期段階であり、今後構想の妥当性や処理の効率性との両立といった課題を明らかにしていく。

プログラミングを用いた音楽制作は近年、Algorave[10] のようなダンスミュージックをコーディングする演奏のように、ポピュラー音楽文化との接近しつつもある。TidalCycles のリズムパターン生成のように、ある演奏スタイルに特化した言語も登場している。一方で現状ポピュラー音楽の制作の中心となっている Digital Audio Workstation ソフトウェア（DAW）は、Ableton Live の中で Max が使用できる Max for Live[11] や、Reaper の拡張機能を作れる ReaScript[12]、Ardour の Lua Scripting[13] など拡張機能としてプログラミング環境を用意しているものはあるとはいえ、例えば音楽の構造のなかに現れる繰り返しや条件分岐といったプログラミング的表現を直接記述することを前提にして作られているものは無く、両者の表現には未だ溝があると言える。

ソースコードを楽譜のように扱い、それを GUI で編集するソフトウェアはプログラミングを用いた音楽表現とポピュラー音楽表現のさらなる橋渡しになる可能性を秘めている。

4. 謝辞

本研究の一部は、日本学術振興会科研費・若手研究 (A) ポストデジタル以降の音を生み出す構造の構築 [17H04772] の支援を受け実施された。

参考文献

[1] Wang, G., Cook, P. and Misra, A.: Designing and Implementing the ChucK Programming Language, *Proceedings of the International Computer Music Conference* (2005).

[2] Orlarey, Y., Fober, D. and Letz, S.: Syntactical and semantical aspects of Faust, *Soft Computing*, Vol. 8, No. 9, pp. 623–632 (online), DOI: 10.1007/s00500-004-0388-1 (2004).

[3] Sorensen, A. C.: Extempore: The design, implementation and application of a cyber-physical programming language, Phd thesis, The Australian National University (2018).

[4] Dannenberg, R. B.: Languages for Computer Music, *Frontiers in Digital Humanities*, Vol. 5, p. 26 (online), DOI: 10.3389/fdigh.2018.00026 (2018).

[5] Norilo, V.: Kronos: A Declarative Metaprogramming Language for Digital Signal Processing, *Computer Music Journal*, Vol. 39, pp. 30–48 (online), DOI: 10.1162/COMJ_a_00330 (2015).

[6] Lattner, C. and Adve, V.: LLVM: a compilation framework for lifelong program analysis transformation, *International Symposium on Code Generation and Optimization, 2004. CGO 2004.*, pp. 75–86 (online), DOI: 10.1109/CGO.2004.1281665 (2004).

[7] Brandt, E.: Temporal type constructors for computer music programming, Phd thesis, School of Computer Science, Carnegie Mellon University (2002).

[8] Hempel, B. and Chugh, R.: Semi-Automated SVG Programming via Direct Manipulation, *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, UIST '16, New York, NY, USA, ACM, pp. 379–390 (online), DOI: 10.1145/2984511.2984575 (2016).

[9] Mayer, M., Kuncak, V. and Chugh, R.: Bidirectional Evaluation with Direct Manipulation, *Proc. ACM Program. Lang.*, Vol. 2, No. OOPSLA, pp. 127:1–127:28 (online), DOI: 10.1145/3276497 (2018).
[10] Collins, N. and McLean, A.: Algorave: A Survey of the History, Aesthetics and Technology of Live Performance of Algorithmic Electronic Dance Music, *Proceedings of the International Conference on New Interfaces for Musical Expression*, London, United Kingdom, Goldsmiths, University of London, pp. 355–358 (online), available from ⟨http://www.nime.org/proceedings/2014/nime2014_426.pdf⟩ (2014).
[11] Ableton: Max for Live, available from ⟨https://www.ableton.com/ja/live/max-for-live/⟩ (2019). accessed on 2019-05-27.
[12] REAPER: ReaScript, available from ⟨https://www.reaper.fm/sdk/reascript/reascript.php⟩ (2019). accessed on 2019-05-27.
[13] Ardour: Lua Scripting — The Ardour Manual, available from ⟨http://manual.ardour.org/lua-scripting/⟩. accessed on 2019-05-27.