mimium: 音と音楽のための自己拡張性の高いプログラミング言語の設計と実装

松浦知也 [*] **城一裕** [†]

概要. 本稿では，時間的に離散した制御と同期的信号処理を 1 つの言語にまとめた音楽用プログラミング言語 *mimium* を報告する．*mimium* は，ラムダ計算ベースの設計・実装に，音を出すための 2 つの最小限の意味論：時間指定関数実行および内部状態を持つ関数のセマンティクスを追加する方針を取ることで，音楽のための言語でありながら汎用言語の理論という資産を活用できる言語となることが期待される．

1 背景と問題設定

音楽のためのプログラミング環境は，“音楽のための最低限の抽象化機構を備えた人工言語”[8] であり，であり，Max[11] や Puredata[12]，SuperCollider[8] をはじめ，コンピューター黎明期から継続的に様々な言語が設計されてきた．この最低限の抽象化機構を備えるとは，慣例的には Unit Generator（UGen）：フィルターやオシレーターといった，信号処理の基本要素 [6]，を言語側で予め実装することであった．この意味では音楽のための言語とは，ユーザーが UGen を組み合わせることで音楽やシステムを作るという，いわばライブラリとしての側面が強かった．

しかし近年では，Faust[10], Kronos[9], Soul[16], Vult[13] のような，UGen 自体の記述に特化したドメイン固有言語（DSL），Extempore[14] のような低レイヤーの記述を含めたライブコーディング環境の登場によって，音楽言語の設計とは，**ハードウェアや OS のようなインフラストラクチャに依存しない，時間方向の抽象化機構を含めた意味論を持つ汎用的なプログラミング言語設計**と，その言語上でのライブラリ設計という 2 つの作業に分化しつつある．

本来，プログラミング言語の設計において時間という要素は考慮されないため，汎用言語上で構築された音楽のための DSL は常に OS のタスクスケジューラやオーディオドライバのような実装に依存し可搬性の無いものとなってしまう．だからこそ，実装に依存しない抽象的な意味論づくりを行う音楽のための言語設計には意義があり，極論を言えばその言語でどのような表現が可能になるかといった問題と独立して議論が可能である．

その上で，Faust のような意味論が厳密に定義されている言語はこれまで，1: 信号処理のような定期的に処理が行われるものと異なる，例えば 1 秒に 1 回だけ発生するような時間方向に離散的な処理の記述の表現がその意味論に含まれていない，2: 言語の内部表現が入出力を持つグラフ構造になっており，ラムダ計算のような汎用プログラミング言語との理論と組み合わせることが難しいといった課題がある．

2 *mimium* の設計

以上の背景を踏まえ，信号処理と時間方向に離散的な処理の両方が記述できるプログラミング言語 *mimium*
(***mi**nimal-**m**usical-med**ium***)[1] を設計した [7].

表 1 は mimium と既存の言語の仕様を比較したものである．*mimium* は Faust や Kronos と同様に一般的な言語では組み込みで用意されるフィルターやオシレータといった UGen 相当の部分をその言語上で記述することができ，その実行速度は LLVM[5] を用いて JIT(実行時) コンパイルを用いることで C++ などの低レベル言語を用いた記述と同等にしている．その一方で，Faust が項書き換え系で，Kronos が System-$F\omega$ を用いたコンパイル時ラムダ計算によって入出力を持つグラフのネットワーク構造を形成してから，C++や LLVM-IR のような低次の表現へと変換しているのに対し，*mimium* は内部表現をラムダ計算の意味論を拡張した木構造の中間表現として持ち，それを LLVM-IR へと変換している．

具体的には，Faust には無かった時間方向に離散的な処理を継時再帰（Temporal Recursion）[15] と呼ばれるデザインパターンを用いて実現可能にしている．これは，関数実行に続けて @ **演算子**と，実行したい時間を記述することでスケジューラが特定の論理時間に副作用を持つ関数を実行するというものだ（Listing 1 における 5 行目）．この最小限の意味論の拡張（その実現方法そのものはランタイム任せになるが）を再帰関数と組み合わせることで定期的

[*] 九州大学大学院芸術工学府
[†] 九州大学芸術工学研究院

[1] `https://github.com/mimium-org/mimium`

表 1. 既存の音楽プログラミング言語と mimium の特徴の比較.

	Pd/SC	ChucK	Extempore	Faust	Vult	Kronos	*mimium*
スケジューラ	○	○	○	-	-	○	○
サンプル精度スケジューリング	-	○	○	-	-	○	○
低レベルな UGen の定義	-	○	○	○	○	○	○
DSP コードの JIT コンパイル	-	-	○	○	○	○	○
UGen 内部状態の関数型表現	-	-	-	Graph	λ 計算	Graph	λ 計算

```
notes = [260,293,330,347,391,440,495]
index = 0
fn updateIndex(){
  index = (index+1)%7
  updateIndex()@(now+48000)
}
fn phasor(freq){
  res = self + freq/48000
  return if (res > 1) 0 else res
}
fn dsp(input)->(float,float){
  out = phasor(notes[index])*0.75 +
    phasor(notes[index]*1.5)*0.25
  return (out,out)
}
```

Listing 1. ***mimium*** **の言語利用のサンプル．2 つの鋸歯状波の周波数が 1 秒毎にドレミの音階に従い 1 秒周期で変化する．**

なイベント実行を可能にする．

また *mimium* で信号処理プロセッサは**内部状態を持つ関数**として表現される．これは，信号を時間方向に遅延する delay 関数のような組み込みの状態付き関数と，関数定義内で使える，**その関数の 1 サンプル前における返り値を参照できる予約語** *self* **によるフィードバック接続表現**の組み合わせによって表現される（Listing 1 における *phasor* 関数）．この表現はコンパイル時に純粋な関数と状態変数ツリーの組み合わせに変換される[2]．

3 議論

Faust や Kronos，Vult のような 2000 年以後に開発された言語は，同一の DSP アルゴリズムをオーディオプラグインや Web，ハードウェアといった幅広いプラットフォームで使い回すためのインフラストラクチャ的な役割を持っていた．これは基本的にソースコードを C++などの別のソースコードへと直接変換し，その言語固有のランタイムを持たないためだ．

UGen を主体とする言語でも，単に音楽や楽器作りのツールでないインフラとしての役割を担うものとして，Puredata をライブラリとして用いる libpd[3] やそれを用いた RjDj[19, p54-67] といった iPhone アプリケーション，PdParty[18] といった例があるが，Puredata のランタイムにおけるスケジューラー自体が OS のタスクスケジューラーに依存するため，例えば Faust では可能な OS の無いマイクロコントローラ上での動作は難しい[3]．

mimium はソースコードのほとんどを LLVM IR へ直接変換しつつも，タスクスケジューリングの機能のみがランタイム依存という構造を取っているため Faust 等と Puredata のちょうど中間的位置付けにある．@ 演算子によるスケジューリング自体は OS に依存しない仕組みで動作しているため，OS の無い環境での動作も視野に入れられる．

mimium は言語仕様としてはそれ自体が新しい表現を可能にするわけではない．しかし，ラムダ計算を基礎とした，組み込み UGen というブラックボックスの少ない独立した言語体系を持つことで，対象とする表現は Faust のような言語よりは広く，それでいてコードの可搬性は高い，と特徴づけられる．

したがって，*mimium* は（演繹的な議論にはなるが）単にコンピューター音楽を作るためのツールとしての役割だけでなく，プログラムとしての音楽作品を流通させるための基盤としての機能が期待できる．例えば，ゲームにおけるインタラクティブ音楽 [17] のような，動的に内容が変化する音楽作品，Brian Eno の iPhone アプリケーションとして配布される，終わることのない生成音楽 [4][19, p68-77] のようなソフトウェアとしての音楽のためのインフラストラクチャとして機能する可能性がある．*mimium* は，音楽制作と聴取においてコンピューターが不可避的に関わる今日において，音楽プログラミング言語を音楽制作のための便利な道具としてだけでなく，音

[2] この意味論拡張は mimium の発表 [7] と同じ会議で発表された *W* 計算 [1] というラムダ計算の拡張言語によって今後厳密な形式化が期待できる．

[3] Heavy[2] のような，ソースコードを Pd と全く異なる仕組みで C++に変換する仕組みのような例外はある．

楽を Play=再生=実行=演奏するための共通基盤と捉え直すための試みである．

4 謝辞

本研究は JSPS 科研費 JP19K21615，2019 年度情報処理推進機構 (IPA) 未踏 IT 人材発掘・育成事業，および GitHub でのスポンサーシップ[4]によって支援されている．

参考文献

[1] E. J. G. Arias, P. Jouvelot, S. Ribstein, and D. Desblancs. The W-calculus: A Synchronous Framework for the Verified Modelling of Digital Signal Processing Algorithms. *Proceedings of the 9th ACM SIGPLAN International Workshop on Functional Art, Music, Modelling, and Design*, 12:35–46, aug 2021.

[2] E. Audio. Heavy Compiler Collection(hvcc). `https://github.com/enzienaudio/hvcc`, Last Accessed on 2021-11-16.

[3] P. Brinkmann, P. Kirn, R. Lawler, C. McCormick, M. Roth, and H.-C. Steiner. Embedding pure data with libpd. In *Proceedings of the Pure Data Convention*, 2011.

[4] B. Eno and P. Chilvers. Bloom by Brian Eno and Peter Chilvers — GenerativeMusic.com, 2008. `http://generativemusic.com/bloom.html` , Last Accessed on 2021-11-16.

[5] C. Lattner and V. Adve. LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. In *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-Directed and Runtime Optimization*, CGO '04, p. 75, USA, 2004. IEEE Computer Society.

[6] V. Lazzarini. The Development of Computer Music Programming Systems. *Journal of New Music Research*, 42(1):97–110, 2013.

[7] T. Matsuura and K. Jo. mimium: A Self-Extensible Programming Language for Sound and Music. In *FARM 2021 - Proceedings of the ACM SIGPLAN International Workshop on Functional Art, Music, Modeling, and Design*, 2021.

[8] J. McCartney. Rethinking the computer music language: SuperCollider. *Computer Music Journal*, 26(4):61–68, dec 2002.

[9] V. Norilo. Kronos: A Declarative Metaprogramming Language for Digital Signal Processing. *Computer Music Journal*, 39(4):30–48, 2015.

[10] Y. Orlarey, D. Fober, and S. Letz. Syntactical and semantical aspects of Faust. *Soft Computing*, 8(9):623–632, 2004.

[4] `https://github.com/mimium-org/mimium/#contributors`

[11] M. Puckette. Max at seventeen. *Computer Music Journal*, 26(4):31–43, 2002.

[12] M. S. Puckette. Pure Data. In *International Computer Music Conference Proceedings*. Michigan Publishing, University of Michigan Library, 1997.

[13] L. L. Ruiz. Vult Language. `http://modlfo.github.io/vult/`, 2020. Last Accessed on 2021-11-16.

[14] A. C. Sorensen. *Extempore: The design, implementation and application of a cyber-physical programming language*. PhD thesis, The Australian National University, 2018.

[15] A. Sorensen and H. Gardner. Programming With Time Cyber-physical programming with Impromptu. In *Proceedings of the ACM international conference on Object oriented programming systems languages and applications - OOPSLA '10*, New York, New York, USA, 2010. ACM Press.

[16] J. Storer. SOUL_Overview.md. `https://github.com/soul-lang/SOUL/blob/master/docs/SOUL_Overview.md`, nov 2019. Last Accessed on 2021-11-16.

[17] M. Sweet. *Writing Interactive Music for Video Games: AComposer's Guide(Game Design)*. Addison-Wesley Professional, 2014.

[18] D. Wilcox. PdParty: An iOS Computer Music Platform using libpd. In *Proceedings of the Pure Data Convention*, 2016.

[19] 直生 徳井, 哲久 永野, 智太郎 金子. iPhone × Music iPhone が予言する「いつか音楽と呼ばれるもの」. 翔泳社, 2009.

未来ビジョン

音楽のためのプログラミング言語という分野は1950年代からの長い歴史がある一方でその評価基準や理論的基盤，研究パラダイムがはっきりしない現状がある．その上で論理時間スケジューリングやリアルタイム処理といった音声特有の技術要素，信号処理自体の知識，言語処理系自体の知識といったさまざまな参入障壁があることから（特に国内では）研究や開発に関わる人が増えず，それゆえ議論ができる場所も少ないといった悪循環がある．

しかし，音楽のためのプログラミング言語は，依然重要なトピックである．なぜなら音に関わる情報処理が昨今の機械学習のようにより高次の情報抽出や操作に重きが置かれる一方，より低レイヤにおける音楽表現を考えることはコンピューターにおける時間の概念の抽象化の方法を研究することそのものであるからだ．

筆者はこのような音楽に関わる技術の基幹的要素に着目することで未来の音楽のための技術の異なる可能性を示す研究を音楽情報処理と区別し“音楽土木工学”と呼称している．今後も mimium 開発や，音楽プログラミング言語という領域自体を広めることも含め多くの人を巻き込んで研究を続けていきたい．