プログラマブルな音楽制作ソフトウェアの開発
― mimium：インフラとしての音楽プログラミング言語 ―

1.　背景

音楽に特化したプログラミング言語は、音楽にコンピュータを用いた新しい抽象化と共有の方法をもたらすものとしてコンピュータの発展とともに様々なものが開発されてきた。これまで比較的実験的な音楽での利用が中心だったのに対し、近年では Algorave に代表されるようにポピュラーミュージックの分野にも積極的に利用されるようになるなど、利用の幅の広がりを見せている。

また、汎用プログラミング言語での最適化技術の発達も背景として、これまでは C 言語などの低級言語で書かれており、ユーザにとってブラックボックスにもなっていた高度な信号処理を簡潔に書け、かつ C 言語と同等の実行時性能を持ち、各 OS や Web ブラウザ、マイコンなど様々なプラットフォームで動作させる事ができる Faust や SOUL といった言語も登場している。

2.　目的

本プロジェクトでは、Faust や SOUL のように信号処理を簡潔にフルスクラッチで書けて、高い実行性能を持ち、多くのプラットフォームで動作しつつ、さらに処理の対象を楽譜レベルまで広げ、単体で音楽の記述ができるような新しい音楽プログラミング言語を開発することを目指した。

3.　開発の内容

本プロジェクトでは、音楽プログラミング言語 mimium（Minimal-Musical-medIUM）の設計とその実行環境の開発を行った。mimium は現在 macOS 向けにオープンソースソフトウェアとして Mozilla Public License 2.0 のもと公開されている。

3.1.　信号処理

mimium では“dsp”という名前の関数を作ることで音を出す事が可能になる。以下の例はホワイトノイズを出力するサンプルコードである。

```
//noise.mmm
fn dsp(time:float)->float{
    return random()
}
```

ここで“random()”関数は-1～1 の範囲でランダムな数を返す組み込み関数である。ランタイムはオーディオドライバのクロックを元にこの dsp 関数を 1 秒間に 48000 回呼び、その返り値をオーディオドライバに渡す。

3.2. self によるフィードバックの表現

mimium では“self”というキーワードを用いる事で、関数の最後の返り値を取得できる。以下のコードは 0 から 1 まで単調増加して 0 に戻るのを繰り返すノコギリ波を返すものである。

```
fn phasor(freq:float){
    return (self+freq/48000)%1
}
fn dsp(time:float)->float{
    return phasor(440)
}
```

3.3. 音声ファイル読み込み

mimium では現在組み込み関数“loadwav”と“loadwavsize”を用いることで音声ファイルを配列に読み込む事ができる。以下はオーディオファイルを読み込みループ再生する例である。

```
name = "/Users/tomoya/Music/Chickens.wav"
mysize = loadwavsize(name)
mywave = loadwav(name)
fn seek(speed:float,size:float){
    return (speed+self)%size
}
fn dsp(time:float)->float{
    return mywave[seek(1,mysize)]
}
```

loadwav 関数の返り値は float の配列で、[position]を用いてアクセスすることができる。mimium の数値型は float のみで、配列に小数点でアクセスすると線形補完された値を返す。

3.4. 制御処理

楽譜レベルの処理など、信号処理に比べて遅い処理は、関数を時間指定して実行する事で実現される。

```
ntrigger = 1
fn setN(val:float){
    ntrigger = val
}
fn playN(duration:float)->void{
    setN(1)@now
```

```
    setN(0)@(now+duration)
}
fn Nloop(period:float)->void{
    playN(50)
    nextperiod = if(random()>-0.3) period/2 else period
    Nloop(period)@(now+nextperiod)
}
Nloop(12000)@0
```

関数呼び出しの後に@マークをつける事で、関数と引数は一度タスクキューに登録され、その時間になったらスケジューラーがその関数を実行する。また予約語“now”を用いることで、ランタイムの現在時刻を取得できる。上のサンプルコードでは“playN”が実行されるたびに“ntrigger”を 1 に変更し、時間が“duration”を過ぎると再び 0 に戻す。関数“Nloop”はさらに“playN”を実行した後、自分自身を“period”サンプル後に再起的に呼び出す事で繰り返し“playN”を実行する。

3.5. モジュラーアーキテクチャ

mimium の実行環境の構成を図 1 に示す。

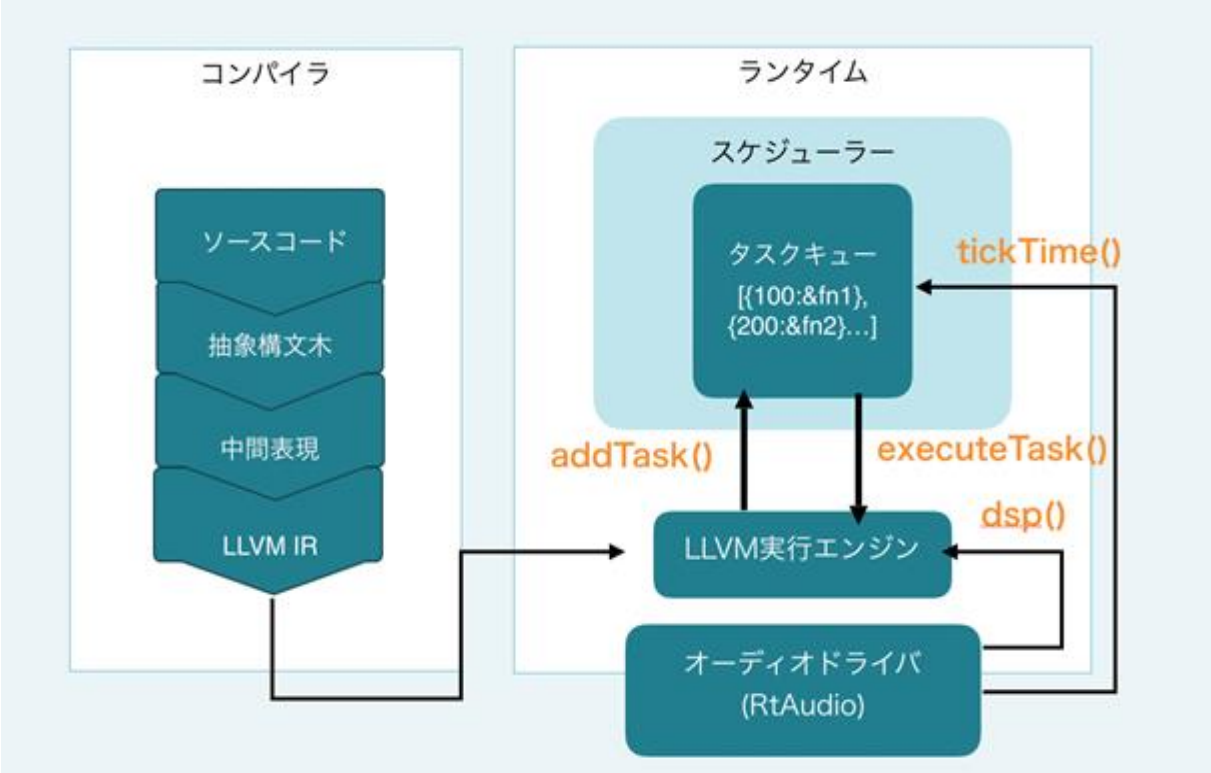

図 1:mimium 実行環境の構成

mimium は、バックエンドに Faust などでも用いられている LLVM というコンパイラ基盤を採用している。LLVM を用いたコンパイラは、LLVM IR という専用の中間表現を介する事で、様々な OS や CPU 向けのネイティブコードを出力する事ができる他、LLVM IR の段階で複

数の最適化を実行し性能をあげる事ができる。さらにネイティブコードをメモリ上でコンパイルし即時実行することも可能である。

mimium の実行環境は主にコンパイラとランタイムの 2 つに分ける事ができる。

コンパイラは、ソースコードを抽象構文木、中間表現といったデータ構造の変換を経て LLVM IR を出力する。

ランタイムは、LLVM IR を受け取り、JIT 実行エンジンを起動し、生成されたネイティブコードを実行する。また、オーディオドライバからのクロックを受け取り、dsp 関数を実行、スケジューラーが時間指定された関数を一度タスクキューに溜め、適切な時間に実行する。

各モジュールは可能な限り分離して運用できるよう設計されているため、コンパイラ部分で LLVM IR を一度ファイルに書き出して、ランタイムがあとで読み込む運用も可能である。

このように各部の独立性を高めることによって、将来的にはオーディオドライバをファイル生成に特化したものに変えて音声ファイルのバッチ処理ツールとしての利用や、オーディオプラグイン上に埋め込む、といった柔軟な運用が可能になる。

4. 従来の技術（または機能）との相違

他の代表的な音楽プログラミング言語と mimium の機能の比較内容を表 1 に示す。

表 1：mimium と他言語の比較表

	Max	SuperCollider	Extempore	Faust	ChucK	mimium
信号(速い)処理	○(Max)	○(hoge.ar())	○(xtlang)	○	○	○
フルスクラッチ信号処理	△(Gen)	×	○	○	△(拡張)	○
制御(遅い)処理	○(MSP)	○(hoge.kr())	○(Scheme)	×	○	○

mimium は、信号処理を低レベルからフルスクラッチで書く事ができ、かつ楽譜レベルの制御処理も記述可能である。Extempore などの言語でも同等の機能を持ってはいるものの信号処理と制御処理に異なる言語処理系を用いており、一つの言語で両方を賄う事ができる事が mimium の特徴である。

5. 期待される効果

mimium を用いる事で、プログラムとしての音楽作品をネイティブアプリケーション、Web アプリケーション、ハードウェアなど様々な環境で稼働させる事が可能になる。音楽を一度 wav ファイルなどに書き出して配布するのではなく、コードのまま配布する事で毎回ランダムに一部が変化する曲や、再生時間が無限の曲のような、プログラムでしか実現し得ない音楽作品の普及を後押しし、新たな音楽表現の可能性を押し広げる効果が期待される。

6.　普及（または活用）の見通し

　ベータ版公開とともに GitHub 上でスターを 100 以上獲得するなど音楽家からもプログラマからも注目を集めている。今後ユーザからのフィードバックを受け、コミュニティを拡大しながら OSS としての利用の幅を広げていく。

7.　クリエータ名（所属）

松浦　知也（九州大学　大学院芸術工学府　芸術工学専攻　博士後期課程）

（参考）関連 URL

開発リポジトリ URL https://github.com/mimium-org/mimium

開発者個人 Web サイト　https://matsuuratomoya.com