2003－MUS－50 (1)
2003／5／16

Audio Engine

リアルタイム MIDI/サウンドレンダリング サーバ

西野　裕樹

慶應義塾大学ＳＦＣ研究所
nishino@earthje.to

本稿では、現在、開発が進行中であるネットワーク透過／マルチ・プラットフォーム対応のMIDI／サウンドレンダリング・エンジンであるAudio Engineについて報告する。このAudio Engineは、現在、同様に開発中のＭＶＣモデルに基づくインタラクティブ 音楽用のフレームワークの一部として設計されている。近年、インタラクティブ音楽の製作に使用されているような単体のアプリケーションと言う形ではなく、サーバとしてインタラクティブ音楽作品制作に必要な機能を簡潔なAPIと共に提供することは、既製品アプリケーションの制約に悩む同分野の作曲家/研究者に対する新たな選択肢となりうるだろう。

Audio Engine

Real time MIDI/Sound Rendering Server

Hiroki NISHINO

SFC Lab. Keio University.

Abstract: This paper describes *Audio Engine*, a new Real-time MIDI/Sound Rendering Server with network transparency and multi platform support currently under development. *Audio Engine* is designed as a part of a MVC architecture-based framework for interactive music systems. Providing such server software with functions and simple APIs required for interactive music works, not as a stand-alone application, can be a new alternative for those composers/researchers of interactive music who are annoyed by a limitation of ready-made software.

1　はじめに

本稿では、現在、製作が進行中のMIDI/音響生成エンジンであるAudio Engineについて報告する。Audio Engineはインタラクティブ音楽アプリケーション製作のために開発されているＭＶＣアーキテクチャに基づくフレームワークの一部であり、リアルタイムのＭＩＤＩおよび音響生成の機能を提供するサーバとして実装されている。

従来、コンピュータ音楽作品用のシステムの製作手法としては、主に大別して以下の２つが主流である。

1） Max/MspやSuper Colliderなどの既成のアプリケーション上で、システムを作成する。

2） 作品に必要な機能をもったソフトウェアをカスタムメイドで作成する（ＳＴＫなどの音響合成用のクラスライブラリが使用されることもある）

このうち、現在、もっとも主流となっているのは1）の既成のソフトウェアを使用してシステムを製作するという手法であろう。しかし、この場合、簡便な開発環境と引き換えに、使用するアプリケーションの持つ制約が、そのまま作品に反映される。

特に、現在インタラクティブ音楽製作環境として、主流となっているであろうＭＡＸのようなビジュアル・パッチングを主体としたソフトウェアでは、この傾向が強いと言える。

つまり、プログラミング経験のない作曲家でも容易にシステム製作が出来るが、このようなビジュアル・パッチング型ソフトウェアの持つ、データ構造やアルゴリズムの表現に弱いという欠点が、そのまま作成されたシステムや作品に反映されてしまう。

この点は、必ずしも作曲された実際の音楽作品の芸術的な価値を減じるものではないが、一方である程度複雑なデータ構造やアルゴリズムの表現を必要とするような作曲ロジックが組み込まれた作品を製作しにくいと言う点では、芸術的／技術的な問題をはらんでいると言える。

また、絶対数は1）に比べて少ないものの2）のように、自身で専用のインタラクティブ・ミュージックシステムを製作する作曲家もいる。この場合ももちろん、使用するプログラミング言語やＯＳなどの環境からの制約は存在するわけではあるが、既成のソフトウェアを使用する場合に比べれば遥かに柔軟である。しかし、リアルタイムのＭＩＤＩ／音響処理という技術的な水準をクリアしている必要があり、その水準は決して低いものではない。既にＳＴＫ[1]などの非常に高機能な音響生成用のライブラリがあるとはいえ、やはり、必要なリソースや時間の管理等はプログラマが行わなければならず、求められる技術的な敷居は一般的な作曲家に対しては非常に高いと言える。

つまり、現状のインタラクティブ音楽の作成状況では、既存のアプリケーションを使用する代わりその制約を音楽的な制約として受け入れるか、もしくは自身で技術的なハードルをクリアしたのちに専用のアプリケーションを作成するという状況にあるといえる。

本稿では、このような状況に対するオルタナティブな可能性として、インタラクティブ音楽システムの製作を容易にするための専用アプリケーション・フレームワークの提案を行い、そのＭＩＤＩ／音響生成に関する部分を担当するサーバーソフトウェアであるAudio Engineについて報告する。

2 Audio Engineの位置付け

Audio Engineが、ＭＶＣアーキテクチャに基づくインタラクティブ音楽用フレームワークの一部であることは既に述べた。このフレームワークの利点とその中でのAudio Engineの位置付けを以下に述べる。

図1のように、このフレームワークではインタラクティブ音楽システムの構成要素を、ＧＵＩ部分をＶＩＥW、ＭＩＤＩ／音響生成を担当する部分をＭＯＤＥＬ、そして作曲上のアルゴリズムを担当する部分をCONTROLとして分割する。Audio Engineはこのうちの MODEL部分の役割を担当する。

一般的に既存のインタラクティブ音楽システム製作用ソフトウェアでは、MIDI/音響処理と作曲アルゴリズムの区別をせず、これらを同一視している場合が多い。しかし、ＭＩＤＩや音響処理に適したunit generatorなどのパラダイムでは、ある程度複雑なデータ構造や手続き的なアルゴリズムなどを扱うことが難しく、この点が従来からのネックになっていた。これはビジュアル・パッチング型のソフトウェアでは顕著である。

このフレームワークではMIDI/音響生成を担当する部分をMODEL、作曲アルゴリズムを担当する部分をCONTROLとして分割することにより、上記の問題の解決を図った。

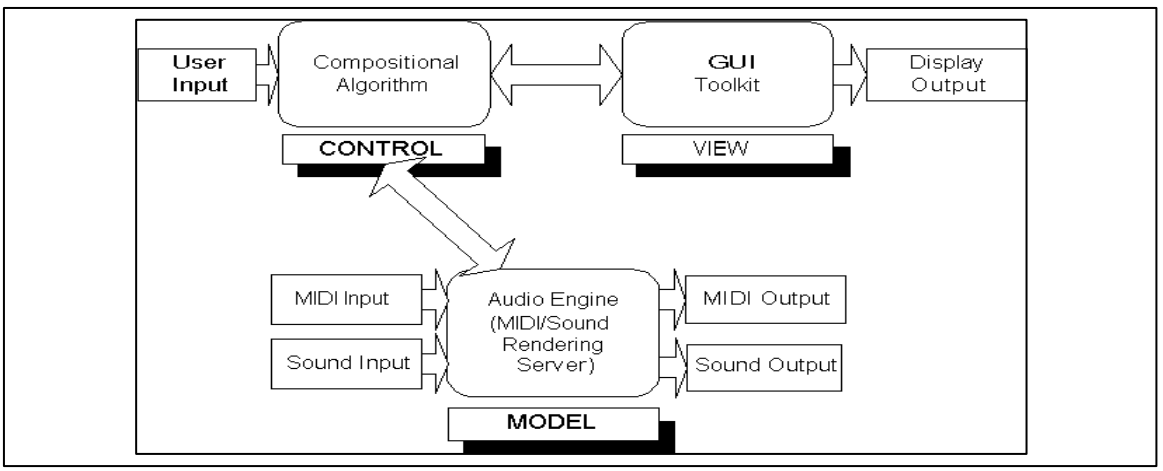

図１：フレームワーク概要

3　Audio Engine機能概要

既に述べられているようにAudio EngineはMIDI/音響生成を担当するサーバーソフトウェアであり、以下のような特徴を実装することを目標にして開発が進んでいる。

MIDI/音響生成用オブジェクトの提供

インタラクティブ音楽システムの音響部分を担当するのに必要なMIDI入出力やオシレータ、フィルタ、サウンドファイル再生/記録用のオブジェクト、簡単な制御を行うための演算オブジェクトなどを提供する

時間／リソース管理

インタラクティブ音楽環境に必要な時間やリソースの管理を行う。これにより、種々のイベントや、その他の時間に関連する処理をプログラミングが簡便になる形で提供する。

マルチ・プラットフォームでの動作

Port Audio／Port MIDIというOS間でのサウンド／MIDI入出力APIの違いを吸収するライブラリを使用し、マルチ・プラットフォーム対応を行う。現状では、この二つのライブラリによってカバーされているLINUX, WINDOWS, MAC OS Xでの動作を目標として開発が行われている。

ネットワーク透過性

Audio　Engineの制御は実際にはTCP/IPソケットを通じて行われる。この際、用意されたAPIによって、ネットワーク通信部を隠蔽し、プログラマはネットワークの存在を意識せずにソフトウェアを作成できる。

マルチクライアントの受け付け

サーバとして同時に複数のクライアントを受け付ける。

４アーキテクチャ概要

図２にAudio Engineのアーキテクチャ概要を示す。

Audio Engineはソフトウェアとしては３層で構成されている。第一層はAudio Engine本体であり、各種MIDI/音響処理や演算に使われるオブジェクト、時間／リソースの管理を行うライブラリが用意されている。ここに記述されているコードは基本的にプラットフォーム非依存となる。

第二層において各プラットフォームにおける際を吸収するためのライブラリが存在する。ここではPORT AUDIO[2], PORT MIDI[3]および独自のTCP/IPラッパクラス群が存在する。ここでは、第三層に存在する各プラットフォーム依存のAPI群をラッピングし、第一層に対してそれを隠蔽する。このようなアーキテクチャにより、Audio Engineをマルチ・プラットフォームに対応させる。

また、Audio Engineの制御は専用のAPIにより隠蔽されたTCP/IP通信によりメッセージを送って行われる。この際プログラマ側からはネットワークを意識せずにAudio Engineを利用可能になっている。

クライアント側プログラムでは、ユニットジェネレータのパラダイムに基づき、MIDI/音響生成用に必要なオブジェクト等の生成を行い、それを互いに接続する必要があるが、その際、実際のオブジェクトは、Audio Engine側に生成される必要があり、そこでクライアントプログラム側の制御にしたがって処理を行う必要がある。つまり、ネットワーク越しにオブジェクトの制御が行われる必要がある。

このような仕組みを実現するに当たって、デザインパターンのremote proxyパターン[4]を簡略化した上で応用している。

remote proxyパターンとは、たとえば、ネットワーク上の別サーバのオブジェクトにアクセスする際に、IPやプロトコルなどの情報を隠蔽してプログラムからは、あたかもローカルなオブジェクトのアクセスしているように扱えるようにするパターンである。

簡略化された点は、ローカルとリモートのオブジェクトが実際の継承関係にはない点と、ローカルで使用されるオブジェクトの種類は１種類のみである点である。

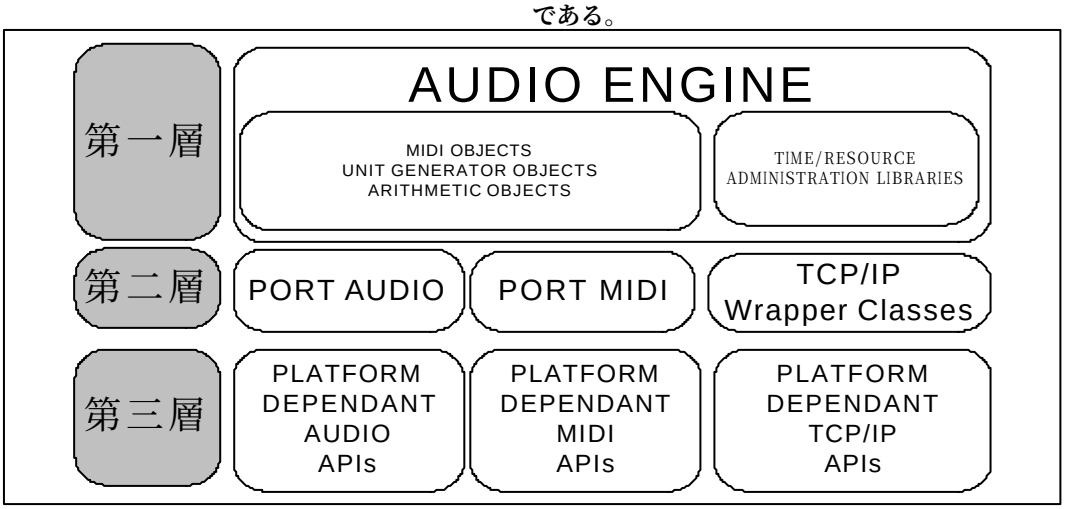

図２：アーキテクチャ概要

継承関係にない理由は、基本的にAudio EngineのAPIはC＋＋以外の多言語対応を念頭においているため、C++以外の言語でクライアントを記述する場合、どのみち継承関係を発生させることができないからである。

また、ローカルでproxyの役割を果たすオブジェクトが一種類である理由は、Audio Engine側で新規オブジェクトが追加された際に、ローカル側で新たに対応するクラスの作成をする必要をなくすためである。

実際にAudio Engine側で動作するオブジェクトを指定するには、ローカル側インスタンスの生成の際に、オブジェクト名を利用して行う。

またオブジェクト間接続なども、ローカルのオブジェクト間にコネクションを張れば、自動的に通信が行われ、リモート（Audio Engine側）でもコネクションが生成される。

以上のような仕組みを通してプログラマ側では、実際にAudio Engine側でオブジェクトを生成／使用する際に、ネットワーク通信を意識せず、クライアント側でオブジェクトを生成するのとほぼ同様の手順で、利用可能にしている。

また、Audio Engine自体はC＋＋で実装されているが、インターフェイスとなるオブジェクトが書かれるべき言語はTCP/IPソケットが使用できる環境であればよいため、JAVAやC#、Cなども使用できる。

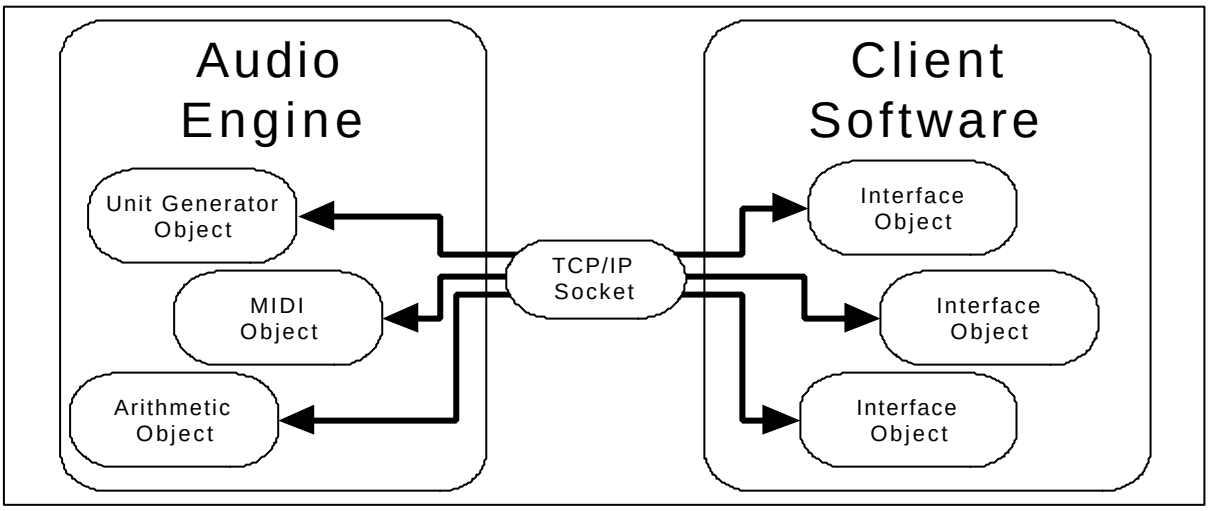

図１：Audio Engineにおけるネットワーク透過性

５コーディング実例

以下にAudio Engineを使用する際の、現在開発中のバージョンでのコーディングの実例を示す。現在のところC++用のAPIのみが用意されている。

このAPIを使用したプログラムの基本的な流れを以下に示す。

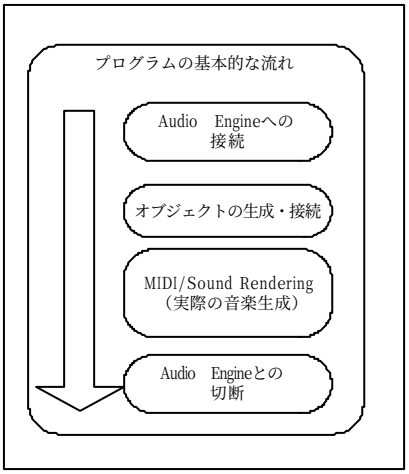

図４：プログラムの流れ

実際のコーディングでは、まず必要なヘッダファイルをインクルードする必要がある。

```
#include “AudioEngineInterface.h”
```

その後、接続するAudio Engineサーバの IPアドレスもしくはホスト名を指定し、サーバに接続する。クライアント側でネットワークを意識するのは、プログラム終了時に、サーバとの接続を切断する時のみである。サーバとの接続には、AEInterfaceクラスのstaticメソッド、connect ()を呼び出せばよい。

以下は、ローカルホスト上のAudio Engineに接続する例である。

```
AEInterface::connect(“127.0.0.1”);
```

あとは特にネットワーク接続を意識する必要はなく、オブジェクトの生成と接続を行っていけばよい。

オブジェクトの生成には、AEInterfaceクラスのstaticメソッド、createObject()に対し、オブジェクト名を渡せばよい。このメソッドは、インターフェイスオブジェクトである、AEObjectのインスタンスへのポインタを返す。

この際、特にMIDIやADC／DAC、UNIT GENERATOR、演算などのオブジェクトの機能種別を

意識する必要はなく、まったく同様に扱ってよい。下の例では、osc~というサイン波を出力するオブジェクトに初期化パラメータ文字列"440"を渡し、周波数の初期値を440Hzにして生成している。次に、dac~オブジェクトという、受け取った信号をサウンド・デバイスから出力するためのDA変換用オブジェクトを生成している。

```
AEObject *pOSC=AEInterface::createObject("osc~","440");
AEObject *pDAC=AEInterface::createObject("dac~" );
```

このメソッドはオーバーロードされており、オブジェクト生成時に渡したいパラメータ文字列や、信号処理が行われる際の優先順位なども指定することが出来る。

生成したオブジェクト間にコネクションを張るには、AEObjectクラスのconnect()メソッドを呼び、接続元のoutletおよび、接続先のオブジェクトおよび、接続先inletを引数に渡せばよい。

```
pOSC->connect(0, pDAC, 0);
pOSC->connect(0, pDAC, 1);
```

この例では先ほど生成したサイン波の出力をDACオブジェクトの０と１の両チャンネルに接続している。

これで、オブジェクトの生成と接続が終わったので、音響生成を行うための準備がととのった。

Audio Engine に DSP を開始させるためには、AEInterfaceクラスのstaticメソッド、startdsp()を呼べばよい。

```
AEIntercace::stardsp();
```

これでDSP処理が開始される。なお、DSP処理が行われている間にもオブジェクトの生成・破棄および接続は動的に自由に行うことが出来る。

また、オブジェクトに対して、パラメータを渡したいときには、inletの番号を指定し、int型，double型，string型などの引数をAEObjectのin()メソッドに渡せばいい。どのinletがどのような引数を受け取るかは、オブジェクトごとに違う。

下の例ではosc~オブジェクトのinlet０番に、int型のパラメータ880を与えている。osc~オブジェクトは、inlet０にintもしくはdoubleの引数を受け取ると、それを現在の周波数として設定する。

```
pOSC->in(0, 880);
```

ここで、MIDI機器からのノート・オンメッセージを周波数として使用したい場合、noteonオブジェクトの出力ををmtofオブジェクトに渡してmidiノートナンバーに変換し、osc~に接続すればよい。

必要のなくなったオブジェクトを明示的に破棄する必要がある場合、AEInterfaceクラスのstaticメソッド、destroyObject()メソッドを呼べばよい。明示的に破棄しない場合は、Audio Engineとの接続が停止するまで存在しつづける。

```
AEInterface::destroyObject(pOSC);
```

ＤＳＰ処理を停止したい場合、AEInterfaceクラスのstopdsp()メソッドを呼ぶことにより停止できる。

```
AEInterface::stopdsp();
```

また、Audio Engineとの接続を終了した場合、AEInterfaceクラスのstatic メソッド、disconnect()を呼べばよい。

```
AEInterface::disconnect();
```

以上が、基本的なプログラミング手順である。リスト１にmidi入力された音程でサイン波の生成を行うプログラムの実例をあげる。

６今後の課題

以下に今後の課題をあげる。

フレームワーク全体の提供

Audio Engine が提供する機能はMIDI/.Sound Rendering関連に限られているため、GUIに関してはの機能は含まれておらず、現在のところは各プラットフォーム上で、それぞれのGUIを使用するほかない。

しかし、インタラクティブ音楽という特性を考えた場合、各OS上で用意されているGUIプログラミング環境が適切であるとは限らない。インタラクティブ音楽システムの要求するlook&feelは、必ずしも一般的なアプリケーションを想定して作られた既存のGUIプログラミング環境と一致するとはいえない。このため、Audio EngineやGUI ToolKitを含むフレームワーク全体の提供を早期に行いたいと考えている。

実際のマルチ・プラットフォーム対応

開発期間などの問題から、複数プラットフォームを念頭に置いた上でのアーキテクチャを取っているにせよ、実際のマルチ・プラットフォームには現在のところまだ対応していない。今後、時期をみて対応してく予定である。

オブジェクトの追加

MIDIや音響処理、演算などのオブジェクトは既に１００種類以上を超えており、基本的な機能はほぼ備えていると言える。特にPure Dataの基本オブジェクト群に対応するオブジェクトは用意済みであるが、今後は、STKライブラリの組み込みなどを通じ、より高レベルな機能を実装していきたいと考えている。

多言語でのＡＰＩの提供

C++だけではなく、JAVAやC#、BASICなど、他の言語に対してもAPIの提供も行う予定である。

７おわりに

本稿では、MIDI/Sound Rendering Serverである Audio Engineについて報告した。設計および（現在進行中である）開発の当初から、より大きなインタラクティブ音楽用のフレームワークの一部として位置付けられているため、Audio Engine単体では、その有用性を理解してもらうのは難しいかもしれない。

とくに、わが国では、自身でプログラムを行うことのできる音楽家はそれほど多くなく、インタラクティブ音楽の分野では、既成のアプリケーションを使用した製作が主流となってしまっている。このため、アプリケーション作成の為のフレームワークの必要性自体が、あまり理解されていない感もある。

しかしながら、既存のアプリケーションに満足しない、特に、はっきりとしたデータ構造やアルゴリズムを必要とするような方向性、言い換えれば作曲ロジックのソフトウェアとしての実装を重要視するインタラクティブ音楽関連の作曲家や開発者にとっては、インタラクティブ音楽に必要なリアルタイムでのＭＩＤＩ／Sound Renderingの機能が、シンプルなＡＰＩとともに提供されることは、作曲家／開発者が作曲ロジックの洗練に専念できるということでもあり、将来的なこの分野の発展に関して、多少の助力にはなるのではないだろうかとも考えている。

参考文献

[1] Cook, P., Synthesis ToolKit in C++ version1.0 SIGGRAPH 1996

[2] Bencina, R., Burk, P. PortAudio- an Open Source Cross Platform Audio API, International Computer Music Conference 2001.

[3]http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/music/web/portmusic/

[4]Gamma E., Helm R., Jhonson R., Vlissides Jhon., オブジェクト指向における再利用の為のデザインパターンP221-232,ソフトバンク 1999

リスト１：Audio　Engineプログラム実例

```
//-------------------------------------------------------Audio Engineプログラム例-----------------------------------------
#include <iostream>
#include <string>
#include “AudioEngineInterface.h”
int main(int argc, char *args[])
{
	//AudioEngineとの接続
	AEInteface::connect(“127.0.0.1”);

	//オブジェクトの生成
	AEObject *pNtIn		=	AEInterface::createObject(	“ntin”	);
	AEObject *pMtoF		=	AEInterface::createObject(	“mtof”	);
	AEObject *pOSC		=	AEInterface::createObject(	“osc~”	);
	AEObject *pDAC		=	AEInterface::createObject(	“dac~”	);

	//オブジェクトの接続
	pNtIn	->connect(1, pMtoF, 0);
	pMtoF	->connect(0, pOSC , 0);
	pOSC	->connect(0, pDAC, 0);
	pOSC	->connect(0, pDAC, 1);

	//dsp処理の開始
	AEInterface::startdsp();

	cout << “何か入力するとdsp処理を終了します” << endl;
	string s;
	cin >> s;

	//dsp処理の終了とAudioEngineとの切断
	AEInterface::stopdsp();
	AEInterface::disconnect();

	return 0;
}
```