音楽情報科学 30－3
（1999．5．29）

LIST/阪大アプリケーションの再構築
－インタラクティブシステム開発用MIDIライブラリの使用－

†上田 健太郎　‡平井 重行　‡片寄 晴弘　†井口 征士
†大阪大学基礎工学研究科システム人間系
‡(財) イメージ情報科学研究所

これまで，幾つかのMacintosh用MIDIアプリケーションではC言語用MIDIライブラリMIDIPascalが用いられてきた．しかし，68Kプロセッサの Macintoshでしか動作しないという問題や，一旦出力バッファに書き込んだMIDIイベントの変更が不可能であるため，セッションシステムなどインタラクティブ性が重要なシステムの開発においてはMIDIデータの入出力スケジューリングに関してプログラミング上制限があるという問題があった．そこで我々はMIDIPascalと互換性を持ち，予めスケジューリングしたMIDIイベントの変更・消去が可能なMac用OMS対応MIDIライブラリを開発した．

本稿では今回実装したライブラリの仕様について述べるとともに，これまでLIST/阪大で開発してきたアプリケーションの再構築やその他システムでの利用を通じて，その有効性を検討する．

Porting LIST/Osaka University's Applications
－ Using of MIDI Library for developing interactive system －

† Kentaro Ueda ‡ Shigeyuki Hirai ‡ Haruhiro Katayose † Seiji Inokuchi
† Faculty of Engineering Science, Osaka University
‡ Laboratories of Image Information Sciense and Technology

MIDIPascal which is a MIDI library have been utilized for development of some MIDI applications on Macintosh. But there are some problems, for instance, it works on only 68K Macintosh, it does not allow change MIDI events that are written into the output buffer once. These problems force us to think about the dealing timing, when we develop the interactive system like a jam session system. Because we can not change or erase scheduled MIDI events. Therefore, we developped a new MIDI libraly which is compatible with MIDIPascal and based on OMS. This library enable us to change and erase scheduled MIDI events.

In this paper, we describe about specifications of the library, porting our 68K Macintosh applications, utilizing in some other interactive systems. Finally, we consider the effectivenes of the library with these works.

1. はじめに

これまで，竹内らのTwoFingerPiano[1][2][3]や青野らのBAND-MASTER[4]，和氣らのJASPER[5]等，LIST/阪大の研究で開発されたアプリケーションはMacintosh上で実装されており，MIDIデータの入出力にはMIDI用ライブラリMIDIPascal[6]を用いていた．MIDIPascalのAPIセットは単純で使いやすく，MIDIドライバとして入出力の手順等をほとんど意識せずにプログラミングが可能という利点があった．

しかし，68KプロセッサのMacintosh用ライブラリであり，既にPowerPCプロセッサに移行したMacintosh（以下PowerMacintosh）ではこれらシステムは動作しないという問題があった．そのため，これら研究成果を最新のマシン環境で動作させるにはMIDIPascal互換でPowerMacintoshで動作するライブラリが必要であった．

また，MIDIPascalの仕様として，一旦出力バッファに書き込んだMIDIデータの変更を行うことはできないため，出力データのスケジューリングに関

してプログラミング上あまり融通がきかないという問題もあった．これは，セッションシステム等のインタラクティブ性が重要なシステムにおいて，一度出力データをスケジューリングしてしまうと，時事刻々と変化する演奏情報に対して即座にレスポンスできないという問題となり，インタラクティブシステムのプログラミングスタイルとして自由度が制限されていたと言える．

そこで，今回我々はこれまでのプログラム資産の活用と，今後の研究におけるプログラミングの自由度の向上を目的とし，上記問題を解決しつつ，MIDIPascal 互換 API セットを持つライブラリ MALIO(Macintosh LIbrary for Oms)を実装した．以下それについて述べると共に，上記アプリケーションの再構築やその他システムの開発を通してライブラリの有効性について考察を行う．

2. MALIO の仕様

MALIO を実装するに当たり，次のような点が要件として挙げられる．

1）MIDIPascal 互換の API セット（表 1 参照）

2）PowerMacintosh 上で動作

3）スケジューリングされた MIDI データへの変更可能

これら要件を考慮した上で，本節では MALIO の仕様と実装に関する内容について述べる．

2.1 PowerMacintosh 対応と OMS の利用

PowerMacintosh対応ライブラリを実現するにあたり，現在のMacintosh用市販MIDIアプリケーションについて考慮すると，多くのソフトウェアはOpcode Systems社のOMS[7](Open Music System) をMIDI入出力ドライバとして用いた構造となっている．OMS自信は68K Macintosh と PowerMacintosh 両方に対応したMIDI ドライバとして実装されており，同一マシン上でOMS対応MIDIアプリケーションが複数動作してお互いにMIDIデータの送受信が可能になる等，MIDI 入出力に関して様々な処理が可能となっている．しかし，OMS を利用したプログラミングはAPI セットが複雑で誰でも簡単にプログラミングできるわけではないという問題がある．

そこでMALIOでは，OMS を下位MIDI ドライバとして用い，MIDIPascal のAPI を上位インタフェースとなるOMS ラッパーとして実装することで，MIDIPascal の容易さとOMS の多様な機能，PowerMacintosh 対応を実現することにした．下位MIDI ドライバがOMS であるため，MALIO を用いた場合，直接OMS のAPI を呼び出すことも可能である．

しかし，ここで問題となるのは指定時間にMIDI データを出力する動作を行うMIDIPascalのMEvtOut () に対応する関数がOMS のAPI には用意されていない．つまり，OMS内部では出力イベントのイベントスケジューラが存在しておらず，プログラマが自ら時間管理を全て行わなければならないということ

表 1．MIDIPascal の代表的な API

int initialize(void)	MIDIPascalを初期化する．
void OutCount(unsigned int *Count)	未出力の出力バッファのデータのバイト数をCountに返す．
void Midi(int Arg)	MIDIの入出力に関する種々の命令を行う． Arg==5　clear input buffer Arg==6　clear output buffer
void MidiTime(unsigned int Ticklength)	MIDIPascalのタイマーの単位を設定する． $\text{TickLength} = \frac{((782000\ \text{ticklength per secs})*(60\ \text{secs per minute}))}{((\text{ticks per beat})*(\text{beats per minute}))}$
void SetTempo(float *BPM, int TicksPerBeat)	テンポと４分音符当たりの分解能を設定する．
void MidiStopTime(void)	MIDIPascalのタイマーを止める．
void MidiStartTime(void)	MIDIPascalのタイマーをスタートさせる．
void MidiGetTime(long * timestamp)	現在の時間をtimestampに返す．
void MidiNow(int Statusbyte, int Databyte1, int Databyte2, int * Result)	すぐにMIDIデータを出力する．
int MEvtIn(int *StatusByte, int *Data1, int *Data2, long *EvtTime)	入力バッファに蓄積されたMDIデータを１つ取り出す． return0　not receive return status (received)
int MEvtOut(int StatusByte, int Data1, int Data2, long EvtTime)	指定時間にMIDIデータを出力する． return0　false return1　true
void QuitMidi(void)	MIDIPascalを終了する．

になる．これに対しては，MALIO内部で用意されたイベントバッファに対し，時間指定された出力データを書き込み，同じくMALIO内部で動作するイベントスケジューラによってMEvtOut()に相当する処理実現した．この内部バッファに書き込まれたMIDIデータの出力情報はMacintoshのTImeManager[8]を用いて時間の管理を行い，指定時間に割り込み処理でMIDIデータの出力を行うようにした．

これにより，ある程度先のイベントまであらかじめスケジューリングしておける機能が実現されたこととなる．

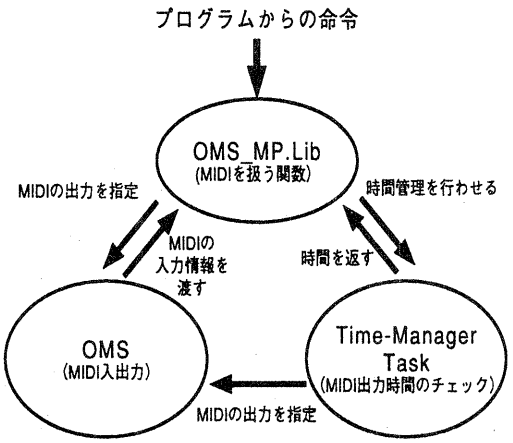

図1．MALIOのアーキテクチャ

2.2 スケジューリングされた出力データの変更

MIDIPascalにおいては，出力データの時間を指定し，一旦バッファにその情報を書き込むと変更することは不可能であった．そのため，出力情報の決定にはスケジューリングを行う特定の時間帯までデータ入力の処理を行い，その直後に一定区間のすべての出力データのスケジューリングを行うといった，処理のタイミングとして負荷が特定の時間帯に集中するプログラミングしかできなかった．結局，入力が終了する時間と出力を行う時間を極限まで切り詰めることによって出力を決定する時間を確保してきたというのが現状である．

例えばJASPERの場合，1小節ごとの出力を，それまでの入力と1小節前の入力を参考にして小節切り替わり直前に決定するといった実装がされたセッションシステムであったため，小節区切りのタイミングに処理負荷のほとんどが集中するシステムであった．このため，特定時間内で可能な処理量に限界が生じ，より複雑で音楽的な解釈を行うシステムに拡張させようとしても，実装上の制限から拡張できないという問題が生じていた．

この問題に加えて，決定された出力情報の変更ができなかったため，インタラクティブ性を重視するセッションシステムにおいて，その時々の情報を有効に活用できず，常に一小節後でしかレスポンスがなかったという問題点も抱えていた．（図2参照）

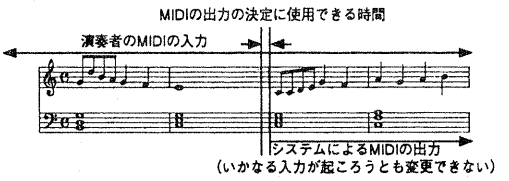

図2．従来のプログラミング方法

そこで，出力情報をバッファに書き込んだ後でも，出力が行われるまでバッファの出力情報を消去もしくは変更可能なAPIを用意できれば，出力を最終的に決定する十分な入力を待つことなく，仮の出力を決定することができ，入力が完了して出力が行われる短時間の間に全ての計算を行わせる必要がなくなる．これにより，セッションシステムなどのプログラミングスタイルとして，音楽の時間の流れの中でいつでも好きな時に出力のスケジューリングができ，やり直しができる他，非常に短いスパンでのレスポンスの実現も可能となる．（図3参照）

MALIOでは以上のような機能を出力バッファと命令の間を取り持つ出力イベントスケジューラを設定することにより実現している．

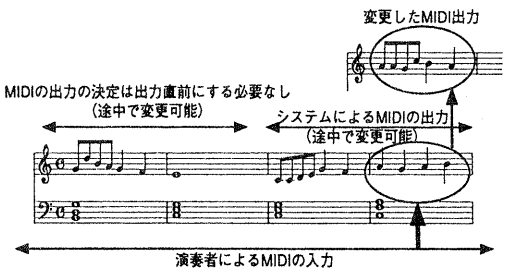

図3．MALIOを用いたプログラミング方法

3. 主要なAPIの機能

本節では特にMALIOの中で主なAPIの機能とその説明を行う．その他APIや，MIDIPascalのAPIとの対応等については付録の表を参考のこと．

short Initialize(short Out_Type)

プログラマはMIDIを扱う種々の命令を行う前にこの関数を呼び出す必要がある．Out_Typeはアプリケーションの性質によって使い分ける．（実際，TwoFingerPianoはMidiNowだけで動作する上，JASPERもMEvtOutだけで動作する．）MIDIPascalで使われていたMidiUtilInit()での入力バッファと出力バッファのサイズの指定はヘッダ内の定義で行う．

long MEvtOut(int status,int note,
int velocity,long timestamp)

MIDIPascalでも設定されていた関数だが，MIDIPascalではバッファに書き込んだ順で出力されるため，時間的に前である出力が後に出されるという問題が生じていた．今回のライブラリではスケジューラに命令を書き込む関数であるが，その際に時間的にソーティングを行っているため，パートごとの出力を決定しその都度スケジューラに書き込んだり，出力を決定した後でも出力を追加する事も可能である．

long MEvtOut2(int status,int note,
int velocity,long timestamp)

上記MEvtOut()と似た機能を持つが，スケジューラに命令を書き込んだ後でもその命令を消去したり，変更する可能性のある命令はこの関数によって行う．返値はその命令のIDとなり，消去したり変更したりする際に参照するために用いる．

short MEvtKill(long ID)

上記のMEvOut2()によってスケジューラに書き込まれた出力命令を，返されたIDを指定することで消去する際に用いる．この間数の返値は0のとき成功した場合，1の場合はすでに出力が実行されたあとでスケジューラに出力命令が存在しない場合である．

short MEvtChange(int status,int note,
int velocity,long timestamp,long ID)

MEvtOut2()によってスケジューラに書き込まれた命令を，返されたIDを指定しMIDIデータを入力することにより変更したい場合に用いる．返り値が0の時は変更が成功した場合，返り値が1の場合はすでに出力が実行された後でスケジューラに出力命令が存在しない場合である．

4. 過去のアプリケーションの移植

前節までに述べたMALIOライブラリを用い実際にTwoFingerPianoとJASPERのPowerMacintoshへの移植を行ってみた．コンパイル環境としてはMetrowerks社のCodeWarrior Profession3を用いた．

TwoFingerPiano：

元々ほとんどGUIの存在しないアプリケーションであり，プログラムの大半の部分がMIDIの入出力を制御する部分であったため，PowerMacintoshに移植する際にはソースコードとしてはアプリケーションの初期化部分だけ書き換えるのみで，あとはMIDIPascalのライブラリをMALIOに入れ替えてリンクするだけでビルド可能であった．またPowerMacintosh上で問題なく動作した．

JASPER：

JASPERは演奏情報の画面表示をstdioのコンソールにて行っていたが，現在のところMALIOの動作としてstdioコンソールと相性が悪いため，演奏情報の画面表示をダイアログに変更し，アプリケーションの初期化部分のソースコードを変更した程度でビルド可能であった．動作としても問題はなかった．

これらの結果から，MIDIPascalからMALIOへのライブラリの移行は，多少ソースコードの変更は必要であったものの，MIDI入出力に関する部分については変更しておらず，スムーズに移植が行えたと言える．

5. 評価

前節で述べた通り，今回実装したライブラリMALIOを用いる事により，我々の過去の研究で開発し，古いマシン環境でしか動作しなかったシステムが最新の環境で動作するようになった．また，MALIOは高柳らの研究[9]や青野らの研究[10]のアプリケーションにおいても活用されている．実際にMALIOを利用した後の彼らの報告によると，

- OMSのプログラミング知識が全くなくとも簡単にMIDIの入出力が行えた．
- スケジューラが出力データのソーティングを行うのでパート毎に出力を生成するプログラミングが気兼ねなく出来る．
- 出力データのダイナミックな変更ができたのでインタラクティブシステム開発用ライブラリとしてかなりの利用価値がある．

という意見が聞けた．

これら報告と前節の移植作業の結果から考慮して，ライブラリの使い易さと，プログラミングスタ

イルの自由度の向上という面においては当初の目的は達成されたと言える．但し，本ライブラリの動作速度に関する評価はまだ行っておらず，どの程度までの出力イベントのスケジューリング変更が可能か，等については今後の課題として残っている．

6. まとめ

本稿においては，MIDIを用いたインタラクティブシステム開発用ライブラリMALIOについてと，ライブラリの実際の利用について述べた．ライブラリの仕様としてはMIDIPascal互換でPowerMacintosh上でも動作可能なもの，MIDI出力イベントのスケジューリングが後から変更可能，というものである．

我々の過去の研究で開発し，最新の環境で動作しなかったシステムをこのライブラリを用いて移植したことや，その他のアプリケーションにおける利用を通して，ライブラリの実装目的は達成されていると言える．特に，プログラミングスタイルの自由度の向上という意味で，これまで処理負荷が一極集中していた計算を時間的に負荷分散させることが可能となり，また計算コストの高い計算も行うことができるようになり，より高度な処理を行うインタラクティブシステムの実現が可能となると考えられる．また，例えばリズムパートなどある程度パターンが決まっている出力をあらかじめイベントスケジューラに書き込み，その時々の処理によりそれらの出力を変更する，といったような処理も可能となり，プログラミングの考え方自体をこれまでと変えることが出来ると考えられる．

今後は，本ライブラリの処理速度に関する実験と共に，その他より様々なMIDIを用いたインタラクティブシステムの構築とその要件に見合ったライブラリの拡張を行っていく予定である．

参考文献

[1] 竹内好弘，片寄晴弘：TwoFingerPianoによる曲想の表現，情報処理学会研究報告，95-MUS-11，pp.37-44 (1995)

[2] 竹内好弘，上符裕一，片寄晴弘，井口征士：TFPの改良と教育現場における評価，情報処理学会研究報告，96-MUS-16, pp.21-25 (1996)

[3] 竹内好弘，上符裕一，片寄晴弘，井口征士：TFPによる邦楽器の伴奏シミュレーションの教育への応用，情報処理学会研究報告，97-MUS-20，pp.1-6 (1997)

[4] 青野裕司，片寄晴弘，井口征士：バンドライクな音楽アシスタントシステムについて，情報処理学会研究報告，94-MUS-8, pp.45-50 (1994)

[5] 和氣早苗，加藤博一，才脇直樹，井口征二；テンション・パラメータを用いた協調型自動伴奏システム：JASPER，情報処理学会論文誌，Vol.35, No.7, pp.1469-1481 (1994)

[6] ALTECH SYSTEMS:MIDIPascal for Macintosh

[7] Opcode Systems,Inc.:OMS(Open Music System)Programming Interface

[8] Apple Computer,Inc.:Inside Macintosh : Processes

[9] 高柳剛：曲の崩れ度をコントロール変数としたセッションシステム，大阪大学大学院基礎工学研究科修士論文 (1999)

[10] 青野裕司，片寄晴弘，井口征士：世代別歌の歌い易さ評価モデルと音楽コンテンツ制作への応用，情報処理学会 インタラクション '99 論文集，pp.67-68 (1999)

付録・MALIO で実装されている API

void Init_Mac_for_OMS(void)	MALIOを用いる場合，MacintoshのToolBoxに関する種々の初期化が必要である．そこで，プログラムのmainの最初でこの関数を呼び出すことにより，OMSを利用するためのToolBoxの初期化を行う．
short Initialize(short Out_Type)	MALIOを起動するための関数で，OMSを起動するとともにMacintoshのTime-Mangerのタスクを設定する． return0 no errar return1 OMS errar return2 OMS input prt errar return3 OMS output port errar return4 OMS open connection errar return5 Time-Manager task errar Out_Type==1 use only Midinow Out-Type==2 use MEvtOut and MEvtOut2 Out_Type==3 use MidiNow,MEvtOut and MEvtOut2
short QuitMidi(void)	MALIOを終了させるときに用いる関数であり，前記のshort Initialize(Out_Type)とセットで用いる． return0 no errar
void Midi(int Arg)	MIDIPascalで設定されていた関数.
void MidiStartTime(void)	MIDIPascalで設定されていた関数.
void MidiStopTime(void)	MIDIPascalで設定されていた関数.
void MidiGetTime(long *time)	MIDIPascalで設定されていた関数.
void MidiTime(unsigned int TickLength)	MIDIPascalで設定されていた関数.
void SetTempo(float *BPM, int TicksPerBeat)	MIDIPascalのMidiUtils.Libで設定されていた関数.
void OutCount(unsigned int *Count)	MIDIPascalで設定されていた関数で，MIDIPascalでは現在の出力バッファにため込まれているMIDIデータのバイト数を返していたが，MALIOでは出力スケジューラにため込まれているMIDIデータのバイト数を返す.
short MEvtIn(int *status,int *Note, int *velocity,long *timestamp)	MIDIPascalで設定されていた関数.
void MidiNow(int status,int note, int velocity,int *result)	MIDIPascalで設定されていた関数.
int MEvtOut(int status,int note, int velocity,long timestamp)	MIDIPascalで設定されていた関数.
long MEvtOut2(int status,int note, int velocity,long timestamp)	MIDIPascalで設定されていたMEvOutで指定時間に出力するMIDIデータを後で消去したり変更したりする可能性がある場合，この命令によって出力を指定する． return ID
short MEvtKill(long ID)	MEvtOut2でスケジューラに書き込んだMIDIデータを消去する.
short MEvtChange(int status,int note, int velocity,long timestamp,long ID)	MEvtOut2でスケジューラに書き込んだMIDIデータを変更する.