社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2008－HCI－128 (17)
2008－MUS－ 75 (17)
2008／5／29

CrestMuseXML (CMX) Toolkit ver. 0.40について

北原 鉄朗　　片寄 晴弘

科学技術振興機構 戦略的創造研究推進事業 CrestMuseプロジェクト／
関西学院大学 理工学研究科

{t.kitahara, katayose}@kwansei.ac.jp

あらまし 本稿では，現在開発を進めているCrestMuseXML Toolkit ver. 0.40について述べる．本バージョンでは，これまで未着手だった音響信号処理用API「Amusa」の整備を主な目的としている．Amusaは，データフロー型の設計になっており，サブモジュールの実装とその接続により信号処理システムを構築する．この方式は，並行処理に適しており，多くの信号処理系タスクを見通し良く実装することができる．

On CrestMuseXML (CMX) Toolkit ver. 0.40

Tetsuro Kitahara　　Haruhiro Katayose

The CrestMuse Project, CREST, JST /
Graduate School of Science and Technology, Kwansei Gakuin University

Abstract This paper describes *CrestMuseXML Toolkit ver. 0.40*, which we have been developing. This version aims to build an API, called *Amusa*, for audio signal processing. Amusa is designed based on the data-flow model where a signal processing system is built by implementing sub-modules and connecting them. This model facilliates concurrent processing and easy development of most signal processing tasks.

1. はじめに

これまでに音楽情報処理に関する様々な研究がなされてきたが，本分野がより一層発展するには，これまでの成果物を再利用したり，シームレスに統合するための枠組みが必要である．このような枠組みがあれば，これまで研究されてきた要素技術を組み合わせて高度な音楽制作支援環境を構築したりというように，既存の研究成果物を最大限に活用した新たな研究を展開することができる．これを実現するには，要素技術ごとに入出力形式が共通化することが不可欠である．そのような考えの下，我々は複数のXMLフォーマットからなるハイブリッドな音楽記述形式「CrestMuseXML」と，CrestMuseXMLを扱うためのオープンソースライブラリ「CrestMuseXML Java Class Library」の開発を進めている☆1)．

これまで，CrestMusePEDB[2]やICMPC-Rencon[3]と連携してプロジェクトを進めていたため，シンボリックな記述方式を中心に開発を進めてきた．楽譜データの記述形式としてMusicXML[4]☆☆を採用し，MusicXMLファイルの読み込み部を実装するとともに，deviation情報（音楽的な表現のための演奏における楽譜からの意図的な逸脱）のための記述形式としてDeviationInstanceXML[1]を独自に設計した．また，スタンダードMIDIファイル（SMF）の読み書きなどにも対応した．その一方，音響信号処理に関しては未着手であった．

現在開発中のver. 0.40では，名称をCrestMuseXML Toolkitに変更し，音響信号処理のためのAPI整備を目標の中心に据えて開発を行っている．音響信号処理では，高速フーリエ変換（FFT）や基本周波数推定，調波構造抽出など，様々な処理モジュールが様々に組み合わされて使用される．これをできるだけ汎用的に実現するため，我々は，Kahn Process Network[5]に基づいたAPIを提供する．Kahn Process Networkは，信号処理のための処理モデルの1種で，プロセスと呼ばれる処理モジュールがチャネルと呼ばれるキューによってつながっており，キューを介してデータがプロセスからプロセスへと順次送られていく．この処理モデルは並行処理にも適しており，信号処理系の多くのタスクを見通し良く実装することができる．

本稿では，**2.** でCrestMuseXMLおよびCrestMuseXML Toolkitの概要を述べた後，**3.** でこの音響信号

☆ http://www.crestmuse.jp/cmx/
☆☆ http://www.recordare.com/xml.html

処理用 API「Amusa」について述べる．4. で今後の展開について述べ，最後に 5. でまとめをする．

2. CMX の概要

CMX は，我々が整備を進めている XML フォーマット群「CrestMuseXML」の略称であり，また，CrestMuseXML を扱うために開発が進められているオープンソースのツールキット「CrestMuseXML Toolkit」を含めたフレームワーク全体を指す場合もある．本稿では，CMX はフレームワーク全体を指すものとし，XML フォーマット群のみを表す場合には CrestMuseXML を用いる．また，ツールキットのみを表す場合は CMX Toolkit と記す．以下，CrestMuseXML と CMX Toolkit の概要について述べる．

2.1 CrestMuseXML とは

CrestMuseXML[1] は，音楽の様々なデータを記述するための XML フォーマット群の総称である．音楽は，波形レベルから楽譜レベル，MIDI レベル，認知構造レベルまで，様々な抽象度の記述が可能である．しかも，認知構造のような高次の表現は，しばしば解釈者によって違いが出てくるため，同一の楽譜データに対して複数の認知構造表現を付与するといった，「1 対多」の記述が求められる．そこで，我々は，あらゆる種類の音楽データを表現できる巨大な XML フォーマットを作るのではなく，抽象度ごとに XML フォーマットを用意し，異なる XML フォーマットに記述された要素間の対応を XLink を使ってハイパーリンクとして表現する，という方針をとった（図 1）．それにより，

- 個々のフォーマットをシンプルに保った上で，複数のフォーマットを組み合わせることで，幅広い種類の音楽記述が可能，
- XLink は「1 対多」のハイパーリンクが可能であるので，同じデータ（たとえば楽譜や演奏データ）に対して複数のメタデータ（楽譜や演奏に対する認知構造的解釈）を付与することが可能，
- ユーザが自分で設計した XML フォーマットを追加することで，記述能力の拡張が可能，
- フォーマットごとにデファクトスタンダードがある場合は，それをそのまま採用可能，

といったメリットが得ることができる．

2008 年 4 月現在では，CrestMuseXML は下記のフォーマットからなっている．

- MusicXML
  Recordare LLC により開発された楽譜データの記述用フォーマット．ノーテーションソフトウェア「Finale」など様々なソフトウェアで対応されている．
- DeviationInstanceXML
  演奏中の楽譜からの逸脱（deviation）情報を記述するための独自フォーマット．MusicXML で表された楽譜データと MIDI XML で表れた演奏データの差分を取ることで生成でき，逆に，DeviationIntanceXML ドキュメントと MusicXML ドキュメントを組み合わせることで MIDI XML ドキュメントを生成することができる．現在のバージョンでは，ピアノ演奏に焦点を当てており，各音符の発音・消音時刻とその強さ，テンポの変動，ペダルに対応している．楽譜には書いてあるけど弾かなかった音や楽譜には書いてないけど弾いた音を記述することもできる．
- MIDI XML
  SMF と 1 対 1 で対応する XML フォーマット．MusicXML DTD ファイル群に含まれており，Recordare LLC が開発したものと思われる．
- SCCXML
  MIDI XML と相互に変換可能なフォーマット☆．SMF や MIDI XML は 1 つの音符が 2 つのメッセージ（NoteOn と NoteOff）に分かれて記述されており，直感的な操作がしにくいので，1 つの音符を 1 つの要素で表された，より単純な記述形式として SCCXML を考案した．一般的な MIDI シーケンサのいわゆるイベントリスト編集画面に基づいた設計になっている．

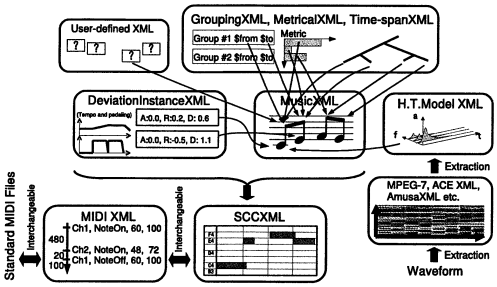

図 1　CrestMuseXML の全体像

2.2 CMX Toolkit

CMX Toolkit は，CrestMuseXML をベースとした音楽情報処理システムを開発するためのフレームワークである．

自動採譜，自動作曲・編曲，演奏の表情づけといった音楽情報処理分野のタスクは，ある抽象度の音楽データを基に異なる抽象度のデータ表現を生成する問題ととらえることができる．つまり，音楽情報処理システムは，一般に (1) 音楽データを入力，(2) 入力された音楽データを処理して異種のデータを生成，(3) 生成された音楽データを出力，という 3 つのステップからなる．これらのステップはこの順でシーケンシャルに行われる場合もあれば，同時並行的に行われる場合もある．データの入出力はファイルを対象とする場合もあれば，MIDI キーボード，MIDI 音源，マイク，スピー

☆ ただし，厳密には，SCCXML ではフォーマットを単純にして直感的な記述をしやすくするため，MIDI XML に比べて記述能力に制限がある．そのため，SCCXML で記述可能なものに一旦変換（正規化）する必要がある．

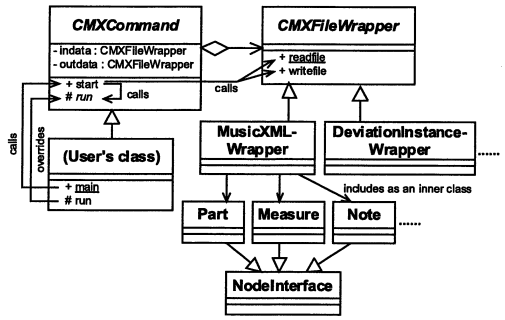

図 2　CMX Toolkit のクラス図（抜粋）

カーといった機器を対象とする場合もある．

2008 年 4 月現在（ver. 0.32）では，ファイルからのデータ入力，ファイルへのデータ出力，シーケンシャルな実行を前提として，ファイルの読み書き部の抽象化と Template Method パターン[6] に基づいたプログラムのスケルトンを提供している．具体的には，以下のクラスが提供されている（図 **2**）．

- CMXFileWrapper クラス
  XML ドキュメントをラップするラッパークラスの共通基底クラス．XML ドキュメントへのアクセスのための API を提供する Document Object Model (DOM) のオブジェクトをフィールドとして持ち，DOM を通じた XML ドキュメントの基本的な入出力 API を提供する．個々の XML フォーマットごとにサブクラスが提供され，MusicXML には MusicXMLWrapper，DeviationInstanceXML には DeviationIntanceWrapper が定義されている．個々の XML フォーマットに特化した入出力 API はこれらのサブクラス内で提供される．
- CMXCommand クラス
  CrestMuseXML を用いた音楽情報処理システムを実装するためのスケルトンを提供する基底クラス．run と start というメソッドが定義されており，start メソッドが呼ばれると，(1) 指定されたファイルを読み込む，(2) run メソッドを呼び出す，(3) 処理結果をファイルに書き込む，という一連の動作が実行される．run メソッドは抽象メソッドとして定義されており，ユーザは，自身のサブクラス内で run メソッドをオーバーライドする．そのため，入力されたデータをどう処理するかだけを run メソッドに記述し，start メソッドを呼び出すコードさえ書けば，プログラムを完成させることができる．

2.3 配布方法

CMX Toolkit は，SourceForge.JP のサービス☆を利用して開発・配布を行っており，BSD ライセンス☆☆に基づいてオープンソースソフトウェアとして配布している．チュートリアルや API リファレンスなどの各種ドキュメントおよびダウンロードサイトへのリンクを http://www.crestmuse.jp/cmx/で提供している．

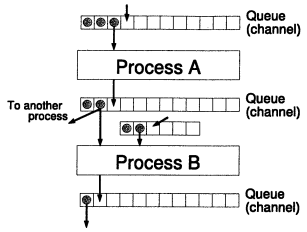

図 3　Kahn Process Network の基本的な考え方

3. 音響信号処理用 API「Amusa」の設計

自動採譜や楽器音認識，楽音の分析・合成といった音響信号処理系タスクでは，通常いくつかの処理の組み合わせで実現される．たとえば，自動採譜であれば，まず，与えられた音響信号に対して短時間フーリエ変換（STFT）やウェーブレット変換などを行って時間周波数表現を得る．その後，周波数成分を調波構造などの制約条件を用いて楽音ごとに群化を行い，さらに，楽器パートごとに群化を行ったり高次の処理を行う[7]．こういった個々のサブモジュールをできるだけ高い独立性を保って実装し，サブモジュールごとの再利用や組換えなどが簡単にできることが望ましい．また，各モジュールの処理が並行に進んでいくモデルが望ましい．

我々は，音響信号処理モジュールの独立性と並行性を確保できる処理モデルとして，Kahn Process Network[5] を採用する．Kahn Process Network は **1.** で述べたように信号処理モデルの 1 種で，プロセスと呼ばれる処理モジュールのネットワークとして信号処理システムを構成する（図 **3**）．各プロセスは，キューから要素を 1 つ取り出して何らかの処理を行い，その結果を別のキューに追加する，という共通の API を持つ．複数のプロセスがキューを介してつながることで，あるプロセスでの処理結果が次々へと後段のプロセスへ送られ，処理が進められる．そのため，並行処理に適したモデルといえる．また，各プロセスが共通の API を持つため，プロセスの再利用や組換えが容易である．

3.1 検討事項

3.1.1 データ構造

Amusa では，データの入力に並行して処理がなされることを想定しているため，データの受け渡しはキューを用いて行われる．キューに格納するデータは原則的に何であっても構わないが，同一キュー内は必ず同種のデータでなければならない．例として多次元ベクト

☆ http://sourceforge.jp/projects/cmx/

☆☆ 「無保証」であることの明記と著作権表示だけを再頒布の条件とするライセンス規定．この条件さえ満たせば，BSD ライセンスのソースコードを複製・改変して作成したオブジェクトコードを，ソースコードを公開せずに頒布できる（Wikipedia より）．

ル，スペクトルピーク集合，MIDI メッセージ，文字列などがあげられる．すべてのデータは時刻の概念を持ち，隣り合うデータ間の時間差が一定の場合（例えば STFT の結果など）と一定でない場合（MIDI メッセージなど）を考える．以下，時系列という場合には前者を指し，後者の場合はイベント列と称することとする．後者の場合は，各イベントの時刻を特定するために，直前のイベントとの時間差（デルタタイム）を取得する手段が必要になる．

並行処理が行われることを想定しなければならないため，あるプロセスがキューからデータ取得を試みたときにデータが未到着だった場合は自動的に到着を待機する機能（ブロッキングキュー）が必要となる．また，通常のキューはデータが取り出されるとキューからは無くなるが，あるプロセスの出力を複数のプロセスが利用する場合が考えられるため，同一データを複数回取り出せなければならない．

3.1.2 API と実装の分離

同じ機能に対して複数の実装を考える場合がある．たとえば，高速フーリエ変換（FFT）の実装を，Java Native Interface（JNI）を介して C 言語で書かれたライブラリを利用して行う高速版と Java のみで書かれた Pure Java 版の 2 種類を提供するといった場合である．前者は高速性が得られる代わりに実行可能環境が制限され，インストール作業が繁雑になるのに対して，後者はインストールは簡単で Java が動作するあらゆる環境で実行可能だが前者に比べて低速である．このようなメリット・デメリットを考慮して容易に実装を選べるよう，API と実装を分離することが必要である．

3.1.3 XML による時系列記述

時系列データを XML で記述するための枠組みとして MPEG-7[8] があるが，仕様が巨大で，手軽に使うにはオーバースペックである感は否めない．そのため，より手軽で単純な汎用の時系列記述形式が望まれる．

なお，ここではイベント列の XML 記述については議論しない．MIDI メッセージなどのイベント列はその内容によって性質が異なると予想されるので，統一的な記述方法を用意するのではなく，内容ごとに（たとえば MIDI メッセージなら MIDI XML というように）記述フォーマットを用意することになるであろう．

3.2 XML フォーマット「AmusaXML」

CMX Toolkit ver.0.40 では，北原がこれまで自身の研究のために開発してきた楽器音認識システム用に用いていたファイルフォーマットを XML 化したもの「AmusaXML」を利用する．現在この楽器音認識システムの CMX Toolkit 上への移植を行っており，その作業を迅速化させるという意図もある．AmusaXML は，**図 4** に示す構造のフォーマットの総称である．ヘッダ部とデータ部に分かれ，ヘッダで STFT の窓幅などのパラメータを記述する．データ部は同一の形式の要素

```
<xxxx>
  <header>
    <meta name="WINDOW_SIZE" content="8192" />
                    ......
  </header>
  <yyyy dim="36" frames="100" timeunit="10">
    zz zz zz zz ......
  </yyyy>
  <yyyy ......>
    zz zz zz zz ......
  </yyyy>
</xxxx>
```

図 4 AmusaXML の基本的な構造

表 1 現在用意されている AmusaXML の具象フォーマット

フォーマット名	トップレベルタグ	データタグ	内容
SPDXML	spd	peaks	スペクトルピーク
TBDXML	tbd	features	音色特徴量
FPDXML	fpd	f0pdf	各 F0 の調波構造の優勢度[9]
IGRAMXML	igram	igram-data	Instrogram[10]（楽器存在確率）

が 1 個以上並び，ここに特徴量などが時系列データとして記述される．元々のファイルフォーマットではバイナリ（IEEE 754 単精度浮動小数点）で記述されているが，XML 版では Base64 でエンコードするかテキストで記述するかになる．トップレベルタグ名とデータ部のタグ名は決まっておらず，これらを内容に応じて適切に決定することで具体的なフォーマットが完成する．現在，**表 1** に示すフォーマットが定義されている．

3.3 クラス設計

Amusa のクラス図を**図 5** に示す．以下，データ構造に関連するクラス，プロセスの定義と実行に関連するクラス，API と実装の分離に関連するクラスの順に述べる．

3.3.1 データ構造に関連するクラス

- AmusaXMLWrapper クラス

  AmusaXML ドキュメントのラッパークラス．CMXFileWrapper クラスを継承した抽象クラスであり，個々のフォーマットごとに具象クラスが用意される（たとえば SPDXML には SPDXMLWrapper）．getDataList メソッドが定義されており，このメソッドを通じて後述の TimeSeriesNodeInterface オブジェクトの配列を取得する．この配列の各要素が AmusaXML ドキュメントのデータ部の各要素に対応する．

- TimeSeriesNodeInterface クラス

  AmusaXML ドキュメントのデータ部の各要素からデータを取得するためのクラス．getQueueReader メソッドで QueueReader オブジェクトが得られるので，その take メソッドを用いることで，時系列データに First-in First-out 方式でアクセスすることができる．

これらのクラスはデータの読み取り専用となっており，直接データを追加することはできない．データの追加を行いたい場合は，次のクラスを利用する．

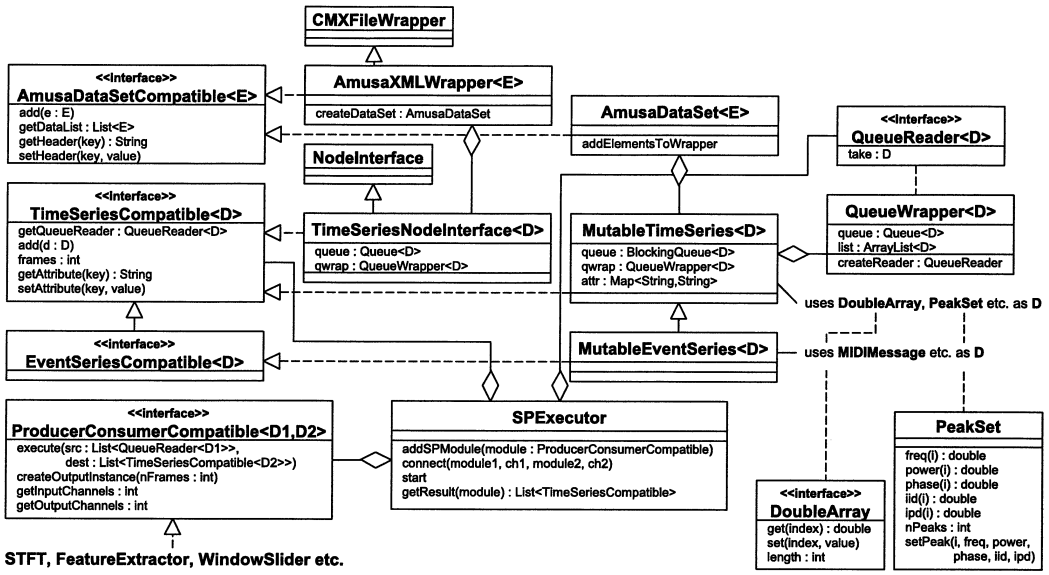

図 5　CMX Toolkit の信号処理用 API「Amusa」の主要部のクラス図

- AmusaDataSet クラス
  AmusaXMLWrapper クラスと同じ API を持つクラスで，要素の追加が可能である．
- MutableTimeSeries クラス
  TimeSeriesNodeInterface クラスと同じ API を持つクラスで，(内部のキューの最後尾に) データを追加することができる．
- MutableEventSeries クラス
  MutableTimeSeries が時系列（時間間隔が一定）を扱うのに対し，イベント列（時間間隔が可変）を扱うためのクラス．

読み取り専用（内部的には DOM の抽象化）と読み書き用（内部的に独自のデータ構造）のクラスを一元的に扱うために，次のインターフェースが用意されている．

- AmusaDataSetCompatible インターフェース
- TimeSeriesCompatible インターフェース
- EventSeriesCompatible インターフェース

TimeSeriesNodeInterface や MutableTimeSeries (そのサブクラスを含む) では，キューを直接扱うのではなく，QueueWrapper クラスでラップし，QueueWrapper によって生成された QueueReader オブジェクトを介してアクセスする．これは，同一のキューから複数回のデータ取得を可能にするためである．QueueWrapper は複数の QueueReader を生成することができる．ある QueueReader がデータ取得を試みたとき，QueueWrapper はキューからデータを取り出すと同時に，そのデータへの参照を配列にも格納する．別の QueueReader がまだそのデータを取得していない場合には，配列からデータを取得して返す．生成されたすべての QueueReader がそのデータを取得した段階で，それへの参照が配列から削除される．これにより，同一データの複数のプロセスへの送信を実現している．

3.3.2　プロセスの定義と実行に関連するクラス

プロセス (データ処理モジュール) は，ProducerConsumerCompatible インターフェースを実装する必要がある．ProducerConsumerCompatible インターフェースには，execute メソッドが宣言されており，ここにそのプロセスが行う処理の内容を記述する．複数のキューからのデータ取得，複数のキューへのデータ出力が可能になっており，取得および出力したいキューの個数を返すように，getInputChannels，getOutputChannels メソッドを実装する必要がある．現在，STFT，FeatureExtractor，WindowSlider などが存在する．

各種プロセスのインスタンスを生成したら，SPExecutor オブジェクトに登録する．たとえば，SPExecutor オブジェクトが `exec` とすると，STFT モジュールとピーク抽出モジュールを以下のようにして登録する．

```
STFT stft = new STFT();
exec.addSPModule(stft);
PeakExtractor peakext = new PeakExtractor();
exec.addSPModule(peakext);
```

次に，登録されたモジュールを接続する．たとえば，STFT モジュールの出力をピーク抽出モジュールに入力したい場合は次のようにする．

```
exec.connect(stft, 0, peakext, 0);
```

最後に，`exec.start();` とすると登録されたプロセスの実行が開始される．

3.3.3　API と実装の分離に関連するクラス

Factory Method パターン[6] を用いて API と実装を

分離する．これは，ある機能を提供するインターフェース P，その実装クラス P'，P のファクトリクラス F と，そのサブクラス F' からなる．F には P の実装オブジェクトを返すメソッド c が抽象メソッドとして定義されており，F' で P' のインスタンスを返すようにオーバーライドされている．当該機能を利用する場合には，F（実際には F'）のインスタンスを取得し，c を呼び出して P（実際には P'）のオブジェクトを得ればよい．P の実装が複数ある場合は各々に対して F' を用意する．F のインスタンスとして具体的にどのサブクラスのインスタンスを取得するかを Java のシステムプロパティで切り替えられるようにすることで，プログラムを変更せずに実装を切り替えることができる．

STFT クラスが利用する FFT などは，この考えに基づいて，FFTFactory クラスと FFT インターフェースを提供している．デフォルトでは Apache Commons Math ライブラリを利用した実装が提供されているが，より高速なものに差し替えることができる．

また，一部の音響信号処理手法については特許の関係上，実装をオープンソースソフトウェアとして公開できないので，同様の方法で API と実装を分離して，API のみを CMX Toolkit に含めている．

3.4 2008 年 4 月現在の実装状況

2008 年 4 月現在，主要な部分についてはほぼ実装は完了している．2008 年 5〜6 月頃の公開を目指して，現在開発を進めている．

4. 今後の展開

本稿で述べた音響信号処理 API は，複数のデータ処理サブモジュールをキューを介して接続するというデータフロー的な考え方に基づいており，並行処理に適したものとなっている．この API の今後の展開として，次のような事柄が考えられる．

- CrestMuse プロジェクト内の各種技術の実装
  CrestMuse プロジェクトでは，優れた音響信号処理技術を数多く所有している．これらを CMX Toolkit 上で再実装することで，自由に組み合わせて利用することが可能になる．現在は，Instrogram 分析[10]，TANDEM-STRAIGHT[11] などが計画されている☆．
- Max/MSP からの利用
  Max/MSP も同様のデータフロー的な設計になっており，Amusa との親和性は高いと思われる．Max/MSP から利用するためのクラスを用意し，Max/MSP から利用した場合のユーザビリティについて検討したい．
- GUI の作成
  各データ処理サブモジュールを配置して接続する過程を GUI でできるようになれば，プログラミングが得意でない人でも簡単に利用できる．こういった GUI の実装を進めていきたい．
- インタラクティブシステムへの適用
  MIDI キーボードやマイクといったデバイスからの入力あるいはデバイスへの出力のための API は，現時点では用意されていないので，これを用意し，インタラクティブなリアルタイム音楽システムへの適用を行う．これを通じて，リアルタイムシステムを実装する上で欠けている機能や API を洗いだし，機能拡充を行っていく．
- 分散処理への適用
  各データ処理モジュールを別々の CPU に割り当てることで，分散処理が可能である．これを実際に試し，分散処理における有用性について検討したい．

☆ ただし，特許などの理由で実装そのものがオープンソースソフトウェアとして公開されるとは限らない．

5. おわりに

本稿では，現在開発を進めている CMX Toolkit ver. 0.40 のうち，特に音響信号処理用 API について述べた．開発が完了し次第，http://www.crestmuse.jp/cmx/ で公開する予定なので，ぜひお使いいただければ幸いである．また，さらなる有用性の向上のため，ご意見をいただければ幸いである．

参考文献

1) 北原鉄朗, 橋田光代, 片寄晴弘: 音楽情報科学のための共通データフォーマットの確立を目指して, 情処研報, 2007-MUS-71, pp. 149–154 (2007).
2) 橋田光代, 松井淑恵, 北原鉄朗, 酒造祐介, 片寄晴弘: 音楽演奏表情データベース CrestMusePEDB ver.1.0 の公開について, 情処研報, 2007-MUS-72, pp. 1–6 (2007).
3) 橋田光代, 片寄晴弘, 平田圭二: Rencon の現状報告と ICMPC-Rencon'08 の実施計画について, 情処研報 (2007).
4) Good, M.: MusicXML: An Internet-Friendly Format for Sheet Music, *The XML 2001 Conf. Proc.* (2001).
5) Kahn, G.: The semantics of a simple language for parallel programming, *Information Processing*, pp. 471–475 (1974).
6) Gamma, E., Helm, R., Johnson, R. and Vlissides, J.: *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Pub. (1995).
7) 片寄晴弘: 自動採譜（概論）, コンピュータと音楽の世界 (長嶋洋一他 (編)), 共立出版, pp. 74–88 (1999).
8) Manjunath, B. S., Salembier, P. and Sikora, T.: *Introduction of MPEG-7*, Jorn Wiley & Sons Ltd. (2002).
9) Goto, M.: A Real-time Music-scene-description System: Predominant-F0 Estimation for Detecting Melody and Bass Lines in Real-world Audio Signals, *Speech Comm.*, **43**, 4, pp. 311–329 (2004).
10) Kitahara, T., Goto, M., Komatani, K., Ogata, T. and Okuno, H. G.: Instrogram: Probabilistic Representation of Instrument Existence for Polyphonic Music", *IPSJ Journal*, **48**, 1, pp. 214–226 (2007). (also published in IPSJ Digital Courier, Vol.3, pp.1–13).
11) Kawahara, H., Morise, M., Takahashi, T., Nisimura, R., Irino, T. and Banno, H.: A temporally stable power spectral representation for periodic signals and applications to interference-free spectrum, F0, and aperiodicity estimation, *Proc. ICASSP*, pp. 3933–3936 (2008).