

CrestMuseXML Toolkit で始める音楽情報処理入門

北原 鉄朗^{†1}

本チュートリアルでは、我々が開発を進めている音楽情報処理研究のためのオープンソースソフトウェア「CrestMuseXML Toolkit」を紹介する。CrestMuseXML Toolkit は、その名の通り、音楽系 XML ドキュメントを扱うためのツールであるが、データフロー型の API も備え、リアルタイム処理を伴う音楽情報処理システムの開発にも利用できる。本チュートリアルでは、XML ドキュメントの処理と音楽データのリアルタイム処理のそれぞれについて、簡単な例題を取り上げて、使い方を解説する。

Introduction to Music Informatics: Get Started with CrestMuseXML Toolkit

TETSURO KITAHARA^{†1}

In this tutorial, we introduce the *CrestMuseXML Toolkit*, our open source software for the research of music informatics. The CrestMuseXML Toolkit is a tool for processing music-related XML documents, as the name implies, but it also has a data-flow API for developing music processing systems that involve real-time processing. In this tutorial, we explain how to use this toolkit based on simple samples of XML document processing and real-time music data processing.

1. はじめに

CrestMuseXML Toolkit は、MusicXML や標準 MIDI ファイル (SMF) の読み書きといったオフライン処理、および、MIDI 入力などに即座に反応するオンライン処理の両方に対応した、音楽情報処理研究のためのオープンソースのライブラリである。

CrestMuse プロジェクトでは、かねてから名ピアノ演奏のアーカイブ化プロジェクトを進めていた¹⁾。これは、名演奏家の演奏内容をオーディオ CD からできるだけ正確に書き下し、演奏に含まれる楽譜からの意図的な逸脱 (deviation と呼ぶ) をアーカイブするというプロジェクトである。この deviation データを MIDI ファイルに埋め込んだ形にするのではなく、楽譜からきちんと分離した形で記述したいという要求があった。そこで、楽譜を表す MusicXML ドキュメントに外からリンクを張る形で deviation データを記述する「DeviationInstanceXML」というフォーマットを考案した。とはいっても単に XML フォーマットを提案するだけでは使ってくれないので、これらの読み書きができるツールを Java で開発することになった。これが CrestMuseXML Toolkit 開発の発端である。

その後、MIDI 信号などをリアルタイムに扱いたいという要求が出てきたため (Max/MSP のような) データフロー型の API 「Amusa」を追加した。この API では、リアルタイム処理全体を「モジュール」と呼ばれる小さな単位に分割し、モジュールに次から次へとデータが流れていくものとする。各モジュールは流れてきたデータを加工し、次のモジュールへ受け渡す。このように、データがモジュールを通るごとに加工されることで目的の処理を達成するというものである。流すデータは、MIDI メッセージでなくても、たとえば音響信号の短い断片や、短時間フーリエ変換の結果でも構わない。そのため、MIDI 処理と音響信号処理を同じ枠組みで実現することができる。さらに、最近では音楽データの確率推論のための API も整備中である。

このように、CrestMuseXML Toolkit は大きく分けて 2 本立ての構造になっている。本チュートリアルでは、各々をどのように使うのか、簡単な例題を通して見ていくこととする。

2. 始める前に—CrestMuseXML Toolkit のインストール—

CrestMuseXML Toolkit は Java で開発されたライブラリであり、実態としてはクラスファイルを固めた jar ファイルである。そのため、jar ファイルを一通りダウンロードして CLASSPATH の通ったところに置けば基本的には完了である。

CrestMuseXML Toolkit のダウンロードページは <http://www.crestmuse.jp/cmxml/> からたどれるので、そこから zip ファイルをダウンロードする。zip ファイルを展開すると jar ファイルが現れるので、それを CLASSPATH の通ったディレクトリに置くか \$JDK_HOME/jre/lib/ext (\$JDK_HOME は Java Development Kit がインストールされているディレクトリを表す) に置く。この辺りの詳細は Java の専門書を参照いただきたい。

CrestMuseXML Toolkit では、XML ドキュメントのパーズなど重要ないくつかの処理

^{†1} JST CREST CrestMuse プロジェクト / 関西学院大学理工学研究科
CrestMuse Project, CREST, JST / Kwansai Gakuin University

を外部ライブラリに頼っている．そのため、これらの jar ファイルも同様にインストールする必要がある．具体的には Apache Xerces, Apache Xalan, Apache Commons-Math が必要である．2009 年 4 月現在ではこれらのライブラリを各 Web サイトから 1 つ 1 つダウンロードするしかないが、本チュートリアル開催日までにリリースを予定している新バージョンでは、必要な jar ファイルをすべて 1 つの zip ファイルにまとめて配布する予定である．

次章にて例題に取りかかる前に、CrestMuseXML Toolkit についていくつかの注意点を述べておきたい．最も重要なことは、単体で便利な機能を有するツールではないということである．あくまで自作のプログラムから機能を呼び出すことを前提としたライブラリである．決して実行したら GUI が出てきて何か便利な処理をクリック 1 つで実行できるというものではない．次に、無保証のオープンソースソフトウェアだということである．たとえば、MusicXML の読み込み 1 つをとっても、決してすべての機能を完全に実装しているのではなく、我々の研究遂行上必要なところから少しずつ実装を進めているというのが現状である．しかも、テストも十分に行っているわけではなく、一応実装はしたけど研究遂行上まだ利用していないのでバグがないか未検証といった部分もちろんある．CrestMuseXML や CrestMuseXML Toolkit の全体像や設計哲学は、文献 2), 3) を参照されたい．

3. 例題 1：簡単な楽譜データに対してごく簡単な表情付けをしてみよう

表情付けとは、与えられた楽譜に対して、それを情緒豊かに演奏した演奏データを生成するというものである．ここではピアノ曲を想定する．ピアノでは、基本的に一度鍵盤を押し込んでしまうと音を制御することはできない．そのため、ここでいう「情緒」というのは、楽譜中の各音符を (1) どのタイミングで (2) どのぐらいの強さで弾いて (3) どのタイミングで鍵盤から指を離すか、だけで実現されているとすることができる．

ここで作成するシステムは、入力は楽譜データである．CrestMuseXML Toolkit では楽譜データとして MusicXML (<http://www.recordare.com/xml.html>) を採用している．出力は MIDI ファイルでもよいのだが、ここでは 1 章で述べた DeviationInstanceXML を用いる．これは大雑把に言うと、MusicXML ドキュメントの各音符に対して、発音時刻をジャストのタイミングからどのぐらいずらすか (attack), 消音時刻を同様にどのぐらいずらすか (release), 音量を基準の音量からどのぐらい変えるか (dynamics) を記述したものである (その他、テンポなど様々な記述仕様がある) ．

ここでは、表情付けのごくごく簡単な第 1 歩として、各音符にスタッカートが付与されれば音の長さを半分にして、それ以外は楽譜通りのタイミングで弾く、音の強さはすべて

同じにする、という例題を取り上げてみる．

CrestMuseXML Toolkit では、基本的に CMXCommand というクラスを継承することで自分独自のコマンドを作成する．たとえばこんな感じである．

```
import jp.crestmuse.cmx.commands.*;
import jp.crestmuse.cmx.filewrappers.*;
public class MyCommand1 extends CMXCommand<MusicXMLWrapper, DeviationInstanceWrapper> {
    protected DeviationInstanceWrapper run(MusicXMLWrapper musicxml) {
        // ここに処理内容を書く ... (*)
    }
    public static void main(String[] args) {
        MyCommand1 c = new MyCommand1();
        try {
            c.start(args);
        } catch (Exception e) {
            c.showErrorMessage(e);
            System.exit(1);
        }
    }
}
```

そうすると、たとえば

```
$java MyCommand1 sampo-solo.xml
```

とすれば sampo-solo.xml を自動的に読み込んで run の中身を自動的に実行する．run メソッド内では、読み込んだ sampo-solo.xml の中身は MusicXMLWrapper というオブジェクトとして取得できるので、MusicXMLWrapper クラスで定義されている様々なメソッドを使って、読み込んだデータの中身にアクセスできる．

MusicXML ドキュメントの中身にアクセスする方法はいくつかあるが、ここでは NoteHandlerAdapterPartwise というクラスを利用する方法を試す．これは音符 (note 要素) ごとと同じ処理を繰り返すときに便利なクラスである．このクラスを用いて各音符に対してスタッカートかどうかをチェックするには、次のようなサブクラスを作成する．

```
import jp.crestmuse.cmx.handlers.*;
class MyHandler extends NoteHandlerAdapterPartwise {
    public void processMusicData(MusicXMLWrapper.MusicData md, MusicXMLWrapper w) {
        if (md instanceof MusicXMLWrapper.Note) {
            MusicXMLWrapper.Note note = (MusicXMLWrapper.Note)md;
            MusicXMLWrapper.Notations notations = note.getFirstNotations();
            if (notations != null && notations.hasArticulation("staccato")) {
                // ここにスタッカートのときの処理を書く ... (**)
            }
        }
    }
}
```

```
}  
}
```

詳細は省略するが、このようなクラスを作成して(*)に

```
musicxml.processNotePartwise(new MyHandler());
```

と書くと、読み込んだ MusicXML ファイルをパースして各音符にスタッカートが付いているかをチェックしてくれる。ために(**)に System.out.println("staccato!"); と書いてコンパイルして実行してみよう。ただし、このままでは run メソッドに return 文がないためにコンパイルエラーになるので、run メソッドの最後に return null; を仮で加えておこう。

次に DeviationInstanceXML の生成・出力を考えてみよう。MusicXML ドキュメントをプログラム上では MusicXMLWrapper というクラスのオブジェクトとして扱ったのと同様に、DeviationInstanceWrapper というクラスのオブジェクトを生成して run メソッドで return すればよい。ただし、諸事情により、DeviationDataSet という別のクラスのオブジェクトを生成し、こちらのオブジェクトに deviation データを一通りセットしてから DeviationInstanceWrapper オブジェクトに変換する。つまり(*)は次のようになる。

```
dds = new DeviationDataSet(musicxml); //DeviationDataSet オブジェクトの生成  
musicxml.processNotePartwise(new MyHandler());  
return dds.toWrapper(); //DeviationInstanceWrapper オブジェクトに変換して return
```

この2行めは MusicXML ドキュメントの各音符に対してスタッカートの有無をチェックして、スタッカートがあれば(**)を呼び出す。(**)では「消音時刻を本来の音長の半分だけ前にずらす」という deviation データを DeviationDataSet オブジェクトに設定すればよい:

```
dds.addNoteDeviation(note, 0.0, -note.actualDuration() / 2.0, 1.0, 1.0);
```

ただし、変数 dds の宣言の場所、MyHandler クラスのインタークラス化など工夫を要するところがある。生成した DeviationInstanceXML を SMF に変換するには、

```
$java jp.crestmuse.cmx.commands.ApplyDeviationInstance -smf <smfname>  
-target <musicxml> <devxml>
```

とすればよい(実際は1行)。

4. 例題 2: MIDI キーボードからの入力をリアルタイムに加工してみよう

まずは、MIDI キーボードから入力された MIDI メッセージをそのまま出力するプログラムから作ってみよう。ただし、ここでは MIDI キーボードのない人のために、PC 用のキーボードを仮想的に MIDI キーボードと見立てて MIDI 入力を行う VirtualKeyboard という

ものを利用する。まずは、次のプログラムを実行してほしい。

```
import jp.crestmuse.cmx.amusaj.sp.*;  
import jp.crestmuse.cmx.sound.*;  
import javax.sound.midi.*;  
public class MyMIDITest {  
    public static void main(String[] args) {  
        try {  
            SPExecutor exec = new SPExecutor(null, 0);  
            MidiInputModule midiin = new MidiInputModule(new VirtualKeyboard());  
            MidiOutputModule midiout = new MidiOutputModule(MidiSystem.getReceiver());  
            exec.addSPModule(midiin);  
            exec.addSPModule(midiout);  
            exec.connect(midiin, 0, midiout, 0);  
            exec.start();  
        } catch (Exception e) {  
            e.printStackTrace();  
            System.exit(1);  
        }  
    }  
}
```

このプログラムでは、MidiInputModule、MidiOutputModule オブジェクトが1章で述べた「モジュール」になっている。SPExecutor クラスはモジュールを管理するためのもので、addSPModule メソッドでモジュールを登録し、connect メソッドで登録されたモジュール同士を接続する。このようにしてモジュールのインスタンス生成、登録、接続が完了したら start メソッドで各モジュールがデータ待ち状態になる。この状態でキーを押すと、MidiInputModule が MIDI メッセージを出力する。すると接続されている MidiOutputModule が MIDI メッセージを受け取り、MIDI デバイスに出力する。

次に、MidiInputModule と MidiOutputModule の間に別のモジュールをはさんでみよう。例として、音高を1オクターブ高くするモジュールを作ってみる。このようなモジュールは次のように書くことができる。

```
import jp.crestmuse.cmx.misc.*;  
import jp.crestmuse.cmx.amusaj.filewrappers.*;  
import java.util.*;  
import javax.sound.midi.*;  
import jp.crestmuse.cmx.amusaj.sp.*;  
class OctaveUp extends SPModule<MidiEventWithTicktime, MidiEventWithTicktime> {  
    public void execute(List<QueueReader<MidiEventWithTicktime>> src,  
        List<TimeSeriesCompatible<MidiEventWithTicktime>> dest)  
        throws InterruptedException {  
        MidiEventWithTicktime me = src.get(0).take();
```

```

ShortMessage sm = (ShortMessage)me.getMessage();
try {
    sm.setMessage(sm.getStatus(), sm.getData1()+12, sm.getData2());
} catch (InvalidMidiDataException e) {
    e.printStackTrace();
}
dest.get(0).add(new MidiEventWithTicktime(sm, me.getTick(), me.music_position));
}
public int getInputChannels() { return 1; }
public int getOutputChannels() { return 1; }
}

```

モジュールの実装は、SPModule クラスを継承することで行う。execute, getInputChannels, getOutputChannels の3つのメソッドをオーバーライドする必要がある。execute メソッドはモジュールの実際の処理内容を書くところである。

execute メソッドの説明をする前に、モジュールにおける「チャンネル」という概念を説明する。「チャンネル」は、あるモジュールの計算結果が2通りあって、片方をモジュール A、もう片方をモジュール B に送りたいときに有用なものである。今回は該当しないので、入力チャンネル数、出力チャンネル数ともに1で、チャンネル番号は0を用いるものとする。

execute メソッドでは、入出力データはチャンネルごとに用意されたキューとして格納される。上記の例では、チャンネル0の入力用キューを取得 (src.get(0)) し、キューからデータを取り出す (take())。このようにして取り出したデータ (MidiEventWithTicktime オブジェクト) に対してノートナンバーをプラス12したものを生成し、チャンネル0の出力用キュー (dest.get(0)) の末尾に追加している (add(...))。

getInputChannels, getOutputChannels メソッドは、入力と出力のチャンネル数を返すようにオーバーライドする。

後は、MyMIDITest クラスを、MidiInputModule と MidiOutputModule の間に OctaveUp モジュールが配置されるように書き換えれば完成である。

CrestMuseXML Toolkit では、上の例の発展形として、MIDI データを再生しながら上記のような処理を行い、MIDI 入力 SMF 上のどのタイミングで起こったかを取得して何らかの処理を行うといったこともできる。また、再生する MIDI データは、SMF としてあらかじめ与えるだけでなく、ユーザの演奏に合わせて動的に生成することもできる。

旋律予測に基づくジャムセッションシステム「BayesianBand」⁴⁾ も、この API を用いて実装されている。この BayesianBand を実装するため、いくつかの新たなクラスを追加した。MusicRepresentation クラスは主旋律、コードといった音楽の記述をベイジアンネット

ワークとして管理するためのものである。たとえば、ユーザが主旋律を弾くと、MusicRepresentation クラスの当該箇所のデータを更新する。データの更新があると、あらかじめ登録された Calculator オブジェクトによって、他のノードの確率分布を自動的に再計算する。この部分は現在開発中であり、仕様が変更される可能性がある。仕様がほぼ固まった段階で、BayesianBand SDK として一般公開する予定である (BayesianBand 自体は公開済)。

5. その他の機能

ここで紹介しきれなかった機能や話題として、次のものがあげられる。

- DeviationInstanceXML におけるテンポなどの記述。
- 音楽の旋律構造を記述する MusicApexXML (2009年4月現在で開発中)。
- 自作 XML フォーマットへの対応。
- STFT, ピーク抽出などの音響信号処理。
- JRuby などを用いた Java 以外の言語からの利用。

6. おわりに

本稿では、チュートリアルで扱う内容に合わせて、CrestMuseXML Toolkit の基本的な使い方を例題を用いて解説した。しかし、紙面の都合からかなり駆け足の説明になってしまい、細部の説明をかなり端折ってしまった。特に Java に慣れていない読者にはわかりにくくなってしまった可能性があるが、本ツールキットの大雑把な仕組み、処理の流れの雰囲気だけでも感じていただければ幸いです。

参考文献

- 1) 橋田光代, 松井淑恵, 北原鉄朗, 片寄晴弘: ピアノ名演奏の演奏表現情報と音楽構造情報を対象とした音楽演奏表情データベース CrestMusePEDB の構築, 情報処理学会論文誌, Vol.50, No.3, pp.1090-1099 (2009).
- 2) 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).
- 3) 北原鉄朗, 片寄晴弘: CrestMuseXML (CMX) Toolkit ver.0.40 について, 情処研報, 2008-MUS-75, pp.95-100 (2008).
- 4) 北原鉄朗, 徳網亮輔, 戸谷直之, 片寄晴弘: BayesianBand: 旋律の予測に基づいた自動伴奏システム, インタラクシオン 2009, pp.31-32 (2009).