DATA-TO-MUSIC API: REAL-TIME DATA-AGNOSTIC SONIFICATION WITH MUSICAL STRUCTURE MODELS

Takahiko Tsuchiya and Jason Freeman

Georgia Institute of Technology
Center For Music Technology
840 McMillan St., Atlanta, GA 30318
takahiko@gatech.edu
jason.freeman@gatech.edu

Lee W. Lerner

Georgia Tech Research Institute
Configurable Computing &
Embedded Systems Laboratory
250 14th Street NW, Atlanta, GA 30318
lee.lerner@gatech.edu

ABSTRACT

In sonification methodologies that aim to represent the underlying data accurately, musical or artistic approaches are often dismissed as being not transparent, likely to distort the data, not generalizable, or not reusable for different data types. Scientific applications for sonification have been, therefore, hesitant to use approaches guided by artistic aesthetics and musical expressivity. All sonifications, however, may have musical effects on listeners, as our trained ears with daily exposure to music tend to naturally distinguish musical and non-musical sound relationships, such as harmony, rhythmic stability, or timbral balance. This study proposes to take advantage of the musical effects of sonification in a systematic manner. Data may be mapped to high-level musical parameters rather than to one-to-one low-level audio parameters. An approach to create models that encapsulate modulatable musical structures is proposed in the context of the new DataToMusic JavaScript API. The API provides an environment for rapid development of data-agnostic sonification applications in a web browser, with a model-based modular musical structure system. The proposed model system is compared to existing sonification frameworks as well as music theory and composition models. Also, issues regarding the distortion of original data, transparency, and reusability of musical models are discussed.

1. INTRODUCTION

Sonification is a unique research field where many applications for scientific as well as artistic purposes coexist. Attempts to define the boundaries and terminologies have been made [1, 2, 3], with widely varying conclusions. Hermann argues that, for instance, for accurate and scientific display of data, sonification is to be separated from a musical approach and expressions, as an artistic painting cannot be regarded as a scientific visualization of data [2].

While this may be true from the perspective of a scientific approach, the creator of a sonification has to contend with the realities of musical cognition and perception [4] and the tendencies of postmodern listening [5].

For example, the use of a continuous or "chromatic" scale of pitch to represent the data points may sometimes accidentally produce harmonically consonant sequences, with small integer multiple relationships, or dissonant harmonies with more distant relationships. Similar concepts may apply to rhythmic alignment, symmetries in phrases, or timbral balance. With daily exposure to music, it is difficult to assume that the listeners' ears are free from such musical perceptions.

This study of a generalized musical sonification framework, and the development of an API[1], explores if it is possible, instead, to utilize these musical effects, theories, techniques, and multi-dimensional structures in data sonification. Musical structures, as long-established forms of "organized sound" [6], have an ability to convey a multitude of information to listeners in a quick and intuitive manner with their hierarchical layers. On the other hand, their complex multi-dimensional nature also raises issues of transparency of data, as well as difficulty of generalizing sonification systems for multiple contexts. We hope to address these issues in the following discussions.

In this article, a new JavaScript web-browser-based API called DataToMusic (DTM) is presented. The API has been in development since the Fall 2014 at the Georgia Tech Center for Music Technology[2]. Our main research interest in this API is to enable simultaneous experimentation, development and application of reusable musical structure models for data sonification.

2. RELATED WORK

2.1. Models and Frameworks

There are several existing frameworks for data sonification. Parameter Mapping Sonification (PMSon) [7] proposes general-purpose methodologies for data analysis, preprocessing, and systematic ways of mapping data to audio synthesis parameters. Model-Based Sonification (MBS) [8, 9, 10] offers a data-agnostic and highly interactive framework, where virtual acoustic objects generated from a data set are manually excited by the user. This method enables users to create models that are "design-once-use-many", unlike PMSon that requires a new mapping specification for each application [8].

In the field of music theory and music technology, various models and generalized frameworks for converting non-musical

[1] Application Program Interface
[2] http://www.gtcmt.gatech.edu/

input to musical outputs have been presented. The late 19th and 20th century composers, such as Schoenberg, Stockhausen, Boulez, and others, investigated methods of numerical and probabilistic manipulations of musical structure in serialism [11, 12]. Contemporary music theorists and scientists, including Toussaint [13, 14] and Tymoczko [15, 16, 17] have also proposed mathematical models to represent and transform musical elements such as rhythm and harmony.

With the introduction of modern technologies, composers explored the possibilities in algorithmic compositions and musical AI systems [18], with rule-based, knowledge-based, and machine-learning-based model structures. An example in this area is David Cope, who proposed models to capture stylistic elements of classical composers in a comprehensive manner [19]. In a recent sonification application, Nikolaidis et al.[20] employed a musical model which maps the visual data of an aquarium to a high-level "tension" parameter of music, where hierarchical rhythm, melody and harmony components were modulated inter-dependently in real-time.

Approaches to the musical structure models are, therefore, diverse and complex. With DTM API, we attempt to implement a generic and abstract structure for variable data input, which may potentially integrate the above-mentioned different musical models for low-level to high-level representation and transformation. The details of the model implementation are discussed in section 3.3.

2.2. Tool Sets and APIs

There have been a number of attempts in the field to provide general-purpose libraries and tool-sets for data sonification. Recent examples of such tool sets include Sonification Sandbox [21], a Java-based cross-platform GUI environment, Interactive Sonification Toolkit developed by Pauletto and Hunt [22] in PureData, and SonART for MacOS [23]. An example of an API is SonData[3] [24], a tool set built for Max/MSP. This API employs both of the previously mentioned sonification frameworks, PMSon and MBS, as modules.

Many sonification applications are also directly built within multipurpose environments such as Max/MSP[4] and Pure Data[5], and real-time sound synthesis environments such as SuperCollider[6] and Csound[7]. These software environments offer immediate or real-time feedback during the development of an application. However, the standalone applications built in such environments may suffer in portability with limited accessibility for various devices, operating systems, and deployability with installation requirements on the user's end. It can also be more difficult to reuse the code, written for specific data sets, in another application without utilizing middle-ware APIs.

It is also worth mentioning data visualization frameworks, particularly browser-based libraries. Protovis [25] is a high-level graph building library which provides a set of building blocks and a system for automatic feature inheritance, scaling and layout in a graph-like data structure. D3.JS [26], the successor to Protovis, has become a popular library for web-based visualization development. It focuses on dynamic mapping of data to low-level HTML elements, enabling development of a wider range of visualization models than its predecessor. RAW[8] is an application and an API built on top of D3.JS. It offers a variety of visualization models that allow the user to quickly map to different data dimensions and configure model parameters.

These visualization libraries have inspired DTM API to offer building blocks in the form of array transformation functions, as well as pre-built instruments that are ready to be used, while offering modularity and reconfigurability.

3. DATA-TO-MUSIC API

DataToMusic (DTM) API is a library for developing data-agnostic sonification programs, and also a real-time environment for experimenting with musical structure models. It was chosen to be a non-GUI-based JavaScript web-browser API for several reasons:

- It may increase the reusability of code, where a GUI-based development tends to limit the integration of the code into different applications, due to, for instance, their inflexibility of code abstraction. (Many objects need to be "visible" or instantiated in order to function.) For a model to be generalized, it needs to be reusable in different contexts.
- It offers a near-zero-cost deployment for the end user, as long as modern web-browsers such as Google Chrome or FireFox are installed. It also makes the application cross-platform including mobile devices.
- Developing a browser-based application makes it easier to integrate to a server application, which may provide streaming data from another web service or host a central database collecting multiple data inputs. It also makes it possible to easily integrate with highly-developed visualization libraries for web browsers, such as D3.JS.
- JavaScript is a dynamic and highly-extensible language which enables on-the-fly extension of model implementations.

On the other hand, graphical programming languages, such as Max/MSP, offer the capability of real-time configuration and interactive development with immediate feedback from the system. This is highly beneficial for developing a real-time audio-based application. Although, for JavaScript, Chrome and Firefox offer a developer console with interactive REPL[9] environment, this may not be sufficient for testing a large-scale application. With text-based audio synthesis languages, such as SuperCollider, an interactive coding paradigm often known as livecoding [27] and JustInTime coding [28] is becoming popular, where part of the code can be selectively re-evaluated without resetting the whole system. DTM API also implements an interface for such use scenarios, which is discussed in a later section.

For its focus on real-time processing of data, along with current browsers' constraints of memory and CPU / thread resources, DTM API focuses more on symbolic processing of information than low-level analysis and audio rendering. It enables the developer to employ high-level musical theories, abstract representations and algorithmic sound composition techniques instead of more prevalent one-to-one and low-level parameter mapping schemes. This allows the playback of high-density and multi-

[3] http://joaomenezes.net/sondata
[4] https://cycling74.com/
[5] http://puredata.info/
[6] http://supercollider.sourceforge.net/
[7] http://csound.github.io/
[8] http://raw.densitydesign.org/
[9] https://developer.chrome.com/devtools/docs/console

dimensional data in musically expressive ways, using WebAudio[10] API as the primary audio rendering engine, with relatively low CPU cost.

In terms of the interface and syntax design, much focus was put on user accessibility and flexibility. Following the successful examples of JQuery[11] and D3.JS, method chaining, often called the "fluent interface"[12], was chosen as the general style of operations. A chained method returns the modified object itself and can be cascaded in sequence, increasing the conciseness and readability of code, and allowing in-line modification of objects.

DTM API aims to provide a low-floor and simple coding interface for casual users and non-expert developers in the audio or music domain. It allows rapid prototyping of applications using the provided general-purpose musical models. With just a few lines of code, the user can load data, instantiate a musical instrument, and map a part of data set to modulate the sound or musical output.

```
// Asynchronously load or query data from local
    or remote location.
dtm.data('sample.csv', sonify);

function sonify(data) {
  var firstCol = data.get('col', 0);
  dtm.instr().pitch(firstCol).play();
}
```

Example 1: DTM Hello World

The mapping of data to the parameters of a musical object is made particularly simple but flexible. With the default *adaptive* mapping mode, the user can feed an arbitrary data format to a musical model, with unknown data size, range, and type, then the model analyzes, rescales and maps the input data to the full range for effective musical expressions. This system is somewhat similar to UrMus, a mobile audio programming environment developed by Essl [29], which enables a simple ad hoc connection between different function "blocks" without a concern of manual scaling.

A simple example of adaptive mapping operation in DTM for a per-note volume modulation model is:

1. If the data type is nominal, convert to a numerical type by taking a histogram.
2. Perform regression analysis of the curvature of the vector.
3. Apply a logarithmic or exponential curve for linear perception of dynamics.
4. Rescale the range from 0.1 to 1 for amplitude multiplication.

These transformation schemes may be modified any time as the models of musical structure can be recreated on the fly. The details of model expansion are discussed in a later section.

The automatic scaling to the full input range can be, however, undesirable when a selected part of data needs to be compared with another, or there is uncertainty for the incoming data stream, as suggested by Pauletto [22]. For this, one may pre-normalize the data according to the known domain value ranges before mapping (*pre-normalized* or *range-preserved* mapping), or choose to do a *literal* mapping by manually transforming the array.

[10]http://webaudio.github.io/web-audio-api/
[11]http://jquery.com/
[12]http://martinfowler.com/bliki/FluentInterface.html

TYPE	METHODS
Generation	fill, clone, reset
Scaling	normalize, rescale, fit, stretch, morph, exp/logCurve, pitchQuantize
Arithmetic	abs, hwr, round, add, mult, powerof, etc.
List Operation	limit, concat, repeat, shift, truncate, block, sort, mirror, invert, shuffle, queue
Nominal	histo, unique, classId, stringify
Unit Conversion	notesToBeats, intervalsToBeats, beatsToIndices, etc.

Table 1: Array Transformation

TYPE	PARAMETERS
List	values, normalized, sorted, uniques, diff, original
Nominal	classes, numClasses, classID, histogram, uniformity
Stats	min, max, mean, mode, median, midrange, std, pstd, var, pvar, sum
Iterator	next, prev, cur, palindrome, random, urn, step, block, blockNext

Table 2: Value Query and Analysis

3.1. Core Modules

The API currently consists of 15 different modules. Among them, the core modules are the **dtm.data** and **dtm.array** for handling data, **dtm.model** and **dtm.instr** for creating modulatable musical objects, **dtm.clock** and **dtm.master** for navigating and reading the data content in a synchronized manner, and **dtm.synth**, **dtm.guido** and **dtm.osc** for audio rendering or other forms of output. The following sections explain each module in detail.

3.2. Data Handling, Analysis and Transformation

The **dtm.data** object is the starting point for sonifying data with this API. It asynchronously loads local files in CSV or JSON format, binary data such as audio or image files, or requests data from web services using REST APIs[13]. It converts the loaded data into JSON key-value format, flattening the nested structure into two dimensional matrix as needed, and stores them in the forms of collections (rows) and arrays (columns).

The **dtm.array** object is the fundamental unit for data handling, which extracts a single dimension, or column, from the JSON collection in the dtm.data object. It is loaded with a large set of analysis and self-transformation functions. The data transformation functions adapt one-dimensional raw data or a potentially unknown data stream to the musical structure models so that they may have meaningful input to generate musical outputs, while retaining the original characteristics of the raw data such as contour, density and value distribution.

Below is a simple example of array transformations. For simplicity, an arbitrary number sequence is used as the source for the

[13]The REST calls services such as Weather Underground (http://www.wunderground.com/weather/api/) and Shodan (http://www.shodanhq.com/), where the response from each service is parsed using the rule dictionaries.

array object.

```
var arrObj = dtm.array([0, 2, -4, 3, 5, -10]);
arrObj.rescale(0, 6);
-> [4, 4.8, 2.4, 5.2, 6, 0]
arrObj.stretch(1.5, 'linear');
-> [4, 4.5, 4.2, 2.7, 3.8, 5.3, 5.8, 3.75, 0]
arrObj.invert(1.0); // With a center point.
-> [-2, -2.5, -2.2, -0.7, -1.8, -3.3, -3.8,
    -1.75, 2]
```

Example 2: Array Transformation

Querying the whole list of values, a single value at certain index, or statistical data is all done through the get(param) method. This is mainly for protecting the data content by cloning the values before returning, as the objects in JavaScript are all mutable. The below example also shows the usage of the array object with streaming data, where the array content is updated with a queue routine.

```
arrObj.getBlock(idx, size).get();
-> [2, 3, 9, 3]
arrObj.get('mean');
-> 4.25
newVal = 7
arrObj.queue(newVal).get();
-> [3, 9, 3, 7]
arrObj.get('mean');
-> 5.5
```

Example 3: Array Value Query

3.3. Musical Structure Models

As described previously, creating a model for describing musical structures is a complex task. Any musical structure is inherently multi-dimensional. For example, a melody played by the violin may have notes with pitch, rhythm and timbre. The pitch may be informed by the tuning, the chosen key and scale, and may imply the current harmony or cadence. The rhythm may consist of the onset time, level, duration, envelope shape, and may be affected by the tempo. The timbre may be informed by the instrument's build, bowing, vibrato, and so on. The melody itself may be a part of a phrase structure which modulates over time. The number of elements in such hierarchical relationships is often much larger, and each of them is to defined or modulated dynamically.

With the development of DTM API, the question of how to generalize such musical structures, in the context of data sonification, is explored. It should ideally take any number and type of data input and return coherent musical results. The approaches taken by the authors are to support automatic or adaptive mappings, and modular and extendable data structures which allows hierarchical and networked information sharing as well as time alignment among the elements. We try to subdivide bigger structures into smaller ones, such as tuning, scale, onsets + duration, and so on. Different musical models share the same data structure, or abstract interface. Models with a similar role may be dynamically swapped, modifying the data mapping scheme and behavior of combined musical output, which is named here as "instrument."

The **dtm.model** is an abstract structure for implementing a unit of dynamic musical structure, such as rhythmicization, note dynamics, articulation, pitch modulation, pitch scale, chord voicing, timbre modulation, and so on. Commonly known musical patterns can also be implemented, such as the Clave rhythm[30], or the II-V-I cadence. With the cadence model, for instance, the first order differential of the array may be used, with which the positive rise triggers the tense dominant chord, and the negative motion triggers the release chord of I.

Models of musical structure can be expanded on the fly with new parameter fields and method names as needed. This is done in a similar way to the JQuery library with its plug-in development system[14].

```
(function (){
  // This stores the new model to the list of
      available models, which can be instantiated
       elsewhere by the name string.
  var m = dtm.model('name', 'categ').register();

  m.mod.pitch = function (arr, mode) {
    if (mode === 'adaptive') {
      m.params.pitch = arr.rescale(60, 100).round
          ().pitchQuantize('major');
    }
    // Return the caller itself. (The parent,
        often an instrument loads this model.)
    return m.parent;
  };
})();
```

Example 4: Model Expansion Example

Each model is loaded as a module into a bigger structure, which can be another model or a **dtm.instr**. Models loaded in a parent model or a **dtm.instr** share information between each other by referring to the parent (caller) and sibling models. The **dtm.instr** object, a collection of smaller musical structure models, outputs the final result as audio or score messages. Each model in the instrument exposes one or more parameters that can be selectively modulated. The instrument object is intended to be a ready-to-play musical device, where the user can load a preset instrument from the instrument collection, connect it to a data source, and immediately hear the musical results. The user may choose to modulate specific parameters of the instrument with data, or the the target parameter can be accessed blindly with a parameter index number. When no parameter is modulated, or mapped to a data source, the musical output is expected as very minimum and static, such as a periodic pulse with a fixed pitch.

```
// Load and start the sound output.
var i = dtm.instr('tumbao').play();

// Modulate a known parameter.
i.intensity(0.3);

// Modulate a parameter by index.
i.modulate(0, arrObj);
```

Example 5: Loading and Modulating a Preset Instrument

Upon encoding non-musical information into a musical structure, there is also a concern of information loss by processes such as quantization, down-sampling, and distortion of data by rescaling or up-sampling. For these problems, a two-layered approach is proposed for manipulating the dynamic musical model, where one may be called static "preparation" and the other dynamic "modulation." In a "preparation" process, one or a series of values are

[14] http://learn.jquery.com/plugins/

mapped instantaneously to shape a characteristic musical motif. For instance, with a specified mapping and transformation routine, a rhythmic pattern may be created from a given sequence of data to signify the characteristic of a particular data column. Furthermore, this may be created from the name of the data column as a character array. A name string "latitude" may be, for example, converted to a sorted class ID list[15], which may look like: [4, 0, 5, 3, 5, 6, 1, 2], then a beat index list: [1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0][16], then fitted to the length of 16 by up-sampling and down-sampling with a step interpolation: [1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1].

This whole sequence of operations may look like:

```
var arrObj = dtm.array(data.get(0).get('key'));
arrObj.classId().intervalsToBeats()
  .fit(16, 'step');
```

Example 6: Shaping / Instantaneous Mapping

Although the content of the pattern shaper can be updated in real-time to gradually shift the shape, this musical motif is not suited for representing the entire characteristics of a particular data sequence by itself. Instead, it may make sense to be used as a signature pattern as part of the ensemble of organized sound.

In contrast to "preparation", with a "modulation" approach, a single value or a small block of an array starting at a certain index can transform the musical material in a continuous manner. For example, a melodic pattern generated from the above-mentioned technique may be enharmonically transposed up and down with a single value or the mean / variance / etc. of a small moving window.

```
var idx = 0; // Index for block-wise reading
dtm.instr().melody(arr)
  .scale(arr.get('block', idx))
  .transpose(arr.get(idx));
idx++;
```

Example 7: Instrument Modulation

The sequential modulation approach is also suitable for multiple dimensions of data to sonify at the aligned read indices. Regardless of the mapping modality, however, both methods of value input accept either a single value or an array.

3.4. Audio Rendering and Other Outputs

Besides loading certain combinations of modules of musical structures, an instrument model may be created with a specific mode of output in mind. Typically, this would be a real-time audio rendering with the **dtm.synth** object, using WebAudio API, or the **dtm.csound** object for Google Chrome-based Csound[17] pNaCl[18] synthesis engine. In the instrument models that are provided in the API, many of the modules harness the unique capabilities that these audio rendering engines offer. For instance, a note-by-note modulation of timbre is possible by generating a new wavetable with a mixture of natural harmonics, utilizing WebAudio's createPeriodicWave() function.

[15]This treats each list item, in this case each character, as a nominal class, and returns numerical IDs.

[16]The interval or note length is converted to the position of index.

[17]http://csound.github.io/

[18]http://vlazzarini.github.io/

```
arrObj.fit(10, 'zeros').normalize().get();
-> [[0.6587, 0.2082, 0.8949, 0.9791, 0.3686,
    0.8929, 0.6555, 0.1797, 0.2698, 0.444]]

dtm.instr().wavetable(arrObj).play();
```

Example 8: Wavetable Synthesis

Other WebAudio-based parameters include note duration, amplitude envelope, 3D binaural panning, comb filter, parametrized convolution reverb and FM synthesis. The classic subtractive synthesis approach may also be used:

```
dtm.instr().voice('noise').lpf(0.3)
  .res(0.5).play();
```

Example 9: Modulating Subtractive Synthesis with Single Values

In this example, the normalized input for the low pass filter is internally scaled logarithmically to set the cutoff frequency in a linearly perceivable manner. These one-to-one parameter mappings are still effective in many situations, and the "default" instrument object exposes them to for quick experimentation.

As alternatives to audio rendering, other forms of output such as Guido musical notation[19] or MIDI messages can be sent through an OSC[20], utilizing the **dtm.guido** and **dtm.osc** objects.

3.5. Real-time Events and Synchronization

The **dtm.clock** object is crucial in the system for real-time data processing, navigation and rendering of audio events. In order to achieve sub-millisecond resolution and adaptation to different environments, such as a server or different browsers, different implementations have been done using a buffer in WebAudio API[21], HTML5 AnimationFrame[22], JavaScript Date object, and NodeJS process.hrtime[23]. In the default state, the clock object uses the AnimationFrame method.

Each clock instance may be either independent or synchronized to the **dtm.master** singleton object. In order to synchronize the clocks with different subdivisions (beats), the master clock counts 480 ticks per beat with a specified tempo, such as 120 BPM, that are referred to by the sub-clocks with different subdivisions. This enables tempo synchronization between different instances of instrument objects. Models within an instrument may also hold synchronized or independent clock instances, which can be accessed to add new callback functions on specified beats.

```
var beats = 8, idx = 0, numVoices = 4;

myModel.on('every', beats, function () {
  // Generate a new chord to be applied to the
      voices
  chord = chordArr.getBlock(idx, numVoices)
    .rescale(0, 11).round().unique().sort();

  idx += beats;
});
```

Example 10: Varied Timing Event Call

[19]http://guidolib.sourceforge.net/

[20]http://opensoundcontrol.org/

[21]https://developer.mozilla.org/en-US/docs/Web/API/AudioBuffer

[22]developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame

[23]http://nodejs.org/api/process.html

4. INTERACTIVE AND REAL-TIME DEVELOPMENT

The process of creating an audio-based application poses a particular challenge to software developers. Unlike near-instant logical operations or static visual elements, the real-time-bound and dynamic audio outputs almost always require in-real-time examination. Adding multi-dimensional musical structures, data navigation or streams, as well as potential inputs from the user introduces further difficulties where the application designer may not be able to predict many possible audio results, simply by not being able to encounter certain combinations of inputs to the system while implementing the audio parameter mapping. In such scenarios, immediate and constant feedback from the system is valuable in the development cycle.

DTM API enables iterative modification of the system such as the expansion of musical structure models on the fly. However, in order to tune the models as the data streams in, or without resetting the data playback to the beginning, there was a need for updating certain parts of the application, such as model, instrument, and data mapping, while retaining the playback of the clocks, value read indices, and already-instantiated objects. The API offers the interface of live-updating the system, with a dtm.clock used for selective evaluations of the system in a beat-synced manner. This run-time reconfiguration is found effective for interactive and iterative development of applications and musical models, where finding effective transformation methods, static parameters, and data source mapping for dynamic models are made easier.

```
// Instantiate objects only once
setup(function () {
  dtm.load('someData', function (data) {
    // Assaign data to a variable
    d = data;
    a = d.get('array', 'someKey');
  });

  // Create an empty model object
  m = dtm.model();

  // Create a default instrument object
  i = dtm.instr();
});

// Set (swap) the model in the rhythm category
    with the new object
i.model(m, 'rhythm').play();

m.mod.sparseness = function (val, mode) {
  // Redefine behaviors on the fly...
};

// Map an array to the model parameter
i.sparseness(a);

m.mod.articulation = function (val) { ... };
```

Example 11: Livecoding in DTM

5. EXAMPLE APPLICATION: SONIFICATION OF ENVIRONMENTAL DATA

An application of DTM API was presented at the Atlanta Science Festival[24] in March, 2015. The Decatur Civic Sonification and Dashboard Project[25] is a collaboration with Georgia Tech Research Institute's Configurable Computing Laboratory[26], where the environmental and traffic data of the city were collected and converted into MIDI and musical scores in real-time. During the event, two custom sensor boxes, each streaming 18 channels of data, were used in combination with DTM API for dynamic remapping, a real-time score system in Max/MSP (Figure 1) and a MIDI rendition in Ableton Live. The score was played by the flutist, cellist and pianist from Sonic Generator[27]. An instrument model for this system was developed, which integrates all the above-mentioned functional requirements.

```
setup(function () {
  fl = dtm.i('decatur').name('Flute');
  vc = dtm.i('decatur').name('Cello');
  pf = dtm.i('decatur').name('Piano');

  d = dtm.data().init(18, 32);
  dtm.osc.on('/rtdata/raw', function(vals){
    d.queue(vals);
  });

  function updateData(c) {
    lux = d.get(0).normalize(2, 12165).limit();
    blue = d.get(1).normalize(10, 21550).limit();
    red = d.get(2).normalize(10, 21550).limit();
    ...
  }

  // 'p' is the ``prescaled'' mapping mode.
  // The mapper also offers ``adapt'' and ``
      literal'' modes.
  fl.pitch(lux, 'p').rep(blue, 'p').ac(red, 'p')
    .play();
});
```

Example 12: Data Streaming and Update

The real-time dynamic mapping of the streamed data was done applying techniques such as range, scaling, block-wise summarization, first order and second order differences, interpolation and extrapolation. The instrument model uses *pre-normalized* rather than the *adaptive* mapping system, where the values are not automatically scaled to the full range, but instead preserves the input dynamic range assuming the data is pre-scaled in 0-1 range.

Given the nature of traditional musical scores, with the relative difficulty of sight reading unanticipated instructions, this instrument model focuses on the use of the most common parameters such as pitch, rhythm and volume. These melodic parameters are presented to the audience with high-level descriptions, such as "variety", "cycle", "articulation", and internally modulate multiple aspects of the instrument (one-to-many mapping). The "variety" parameter, for instance, maps the sequence data values to each available note, changing the duration of individual notes while retaining the combined length of the melodic phrase (Figure 2). When the data moves less actively, the rhythm remains static and constant. When the range of movement increases, the rhythmic pattern becomes more irregular and unique, reflecting the dynamics of the input data.

The combined duration of the melodic phrase is also modified by the "cycle" parameter, which creates a repeated pattern from

[24] http://atlantasciencefestival.org/

[25] http://atlantasciencefestival.org/events/event/1095

[26] http://configlab.gatech.edu/

[27] http://www.sonicgenerator.gatech.edu/

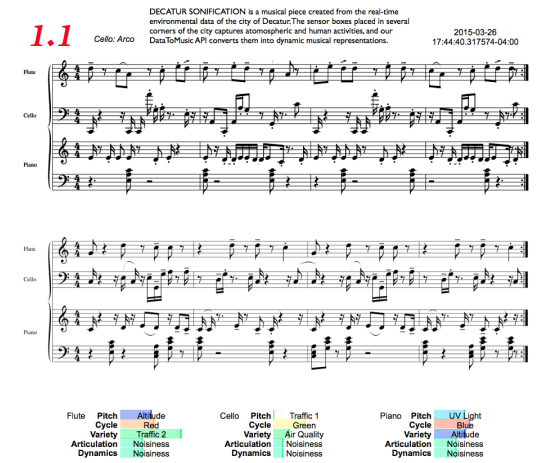

Figure 1: Notation Output in Max/MSP

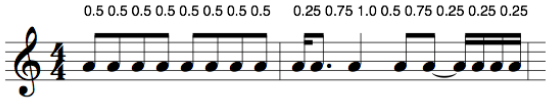

Figure 2: Example of Variety Parameter Modulation

1 to 7 times, according to a summarized value, such as mean or variance, of the data buffer. This also affects the total length of the melodic phrase. The "articulation" and "dynamics" parameters modify each note's attributes, such as sub-note duration, slurs, accents and rests. Finally, the "pitch" parameter simply modulates each note's pitch. During the development, rather than using the absolute pitch positions, fixed motivic melody being transposed by the data was considered. However, as the composition became relatively minimalistic with lots of phrase repetitions, using a motivic shape was not suitable. Also, introducing the "preserved" range mapping added the natural transposition effect which conflicted with the melodic transposition parameter. As a result, the "decatur" instrument focused on the change of dynamics from static to active, generating new melodic shapes every moment, rather than using pre-generated motives. The composition was presented twice in the form of a semi-fixed piece, where the musical form and data mapping were pre-composed, as well as a livecoding piece where the composition was developed as a real-time reaction to the incoming data. The minimalistic style and arrangement chosen for the pieces, inspired by composers such as Steve Reich and Terry Riley, were effective for the form-less and gradually changing musical representation of the data. The phrase modulations also worked effectively when the change of the data frames was moderate, but were sometimes too sudden in change or too static according to the data movement.

This project showed that, using the API, real-time reconfiguration of mapping and data-driven composition were possible even involving human musicians and score generation.

6. FUTURE WORK

Currently, DTM API is in the beta state. A demo application with an interactive web editor and the API documentation has been deployed online[28]. A formal user study and evaluation are being conducted before the public release. The evaluations include the HCI aspect of the programming interface, a computational scalability test, and cognitive and perceptual listening tests, with and without the proposed musical structure models, as well as the adaptive mapping methodologies.

In terms of the technical limitations, firstly, loading of data is capped at around 500 MB with the current web-browsers. In order to handle bigger data sets, the API may need to be integrated to server applications as well. For this, the usage of WebAudio is turned off by default, so that the API may be loaded and used in a server application.

dtm.data currently can perform a minimal range of statistical data analysis and preprocessing. It is planned to add such functions, including dimensionality reduction using PCA, as proposed in PMSon data analysis framework [7].

Lastly, the high-resolution **dtm.clock** successfully enables a large number of event calls, including data handling and audio event triggering, in dynamic tempo at a reliable stability. The current implementations, however, fluctuate with between 1 ms to 10 ms of jitter, due to, for example, a large process buffer size in WebAudio or the limitation of the processing thread in AnimationFrame. This may limit the use of clock for low-level control of audio parameters. A more stable clock implementation combining real-time and scheduling methods will be explored.

7. CONCLUSIONS

Our study of generalizable musical structure models in DTM API, therefore, explores the use of aesthetics and hierarchical layers of sound to represent data in a systematic manner. The API provides an easy prototyping system and an interactive environment for further investigating the models for different musical structures suited for data sonification.

8. REFERENCES

[1] B. N. Walker and M. A. Nees, "Theory of Sonification," 2011. [Online]. Available: http://sonify.psych.gatech.edu/~ben/references/nees_theory_of_sonification.pdf

[2] T. Hermann, "Taxonomy and Definitions for Sonification and Auditory Display," in *Proceedings of the 14th International Conference on Auditory Display (ICAD 2008)*, 2008. [Online]. Available: http://pub.uni-bielefeld.de/publication/2017235

[3] B. Hogg and P. Vickers, "Sonification abstraite/sonification concrete: An 'aesthetic persepctive space' for classifying auditory displays in the ars musica domain," June 2006. [Online]. Available: https://smartech.gatech.edu/handle/1853/50641

[4] D. Deutsch, *Psychology of music*. Elsevier, 2013.

[5] J. D. Kramer, "The nature and origins of musical postmodernism," *Postmodern music/postmodern thought*, vol. 66, pp. 7–20, 2002. [Online].

[28] https://dtmdemo.herokuapp.com/

Available: https://books.google.com/books?hl=en&lr=&id=dtxEAQAAQBAJ&oi=fnd&pg=PA13&dq=The+Nature+and+Origins+of+Musical+Postmodernism&ots=lCRiP0pEN0&sig=p4qhMFnlZGNB0WScalTIhbYyb6U

[6] G. Pape, "Varse the visionary," *Contemporary Music Review*, vol. 23, no. 2, pp. 19–25, 2004. [Online]. Available: http://www.ingentaconnect.com/content/routledg/gcmr/2004/00000023/00000002/art00003

[7] F. Grond and J. Berger, "Parameter mapping sonification," *The sonification handbook*, pp. 363–397, 2011.

[8] T. Hermann, "Model-based sonification," *The Sonification Handbook*, pp. 399–427, 2011.

[9] T. Hermann and H. Ritter, "Listen to your Data: Model-Based Sonification for Data Analysis," in *189194, Int. Inst. for Advanced Studies in System research and cybernetics*, 1999, pp. 189–194.

[10] ——, "Model-based sonification revisited: authors' comments on Hermann and Ritter, ICAD 2002," *ACM Transactions on Applied Perception (TAP)*, vol. 2, no. 4, pp. 559–563, 2005. [Online]. Available: http://dl.acm.org/citation.cfm?id=1101557

[11] R. S. Brindle, *Serial composition*. Oxford University Press, 1969.

[12] R. P. Morgan, *Twentieth-century music: a history of musical style in modern Europe and America*. Norton, 1991.

[13] G. Toussaint, "Computational geometric aspects of rhythm, melody, and voice-leading," *Computational Geometry*, vol. 43, no. 1, pp. 2–22, Jan. 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S092577210900042X

[14] ——, "The Euclidean Algorithm Generates Traditional Musical Rhythms." Bridges Conference, 2005, pp. 47–56. [Online]. Available: http://archive.bridgesmathart.org/2005/bridges2005-47.html

[15] D. Tymoczko, "The Geometry of Musical Chords," *Science*, vol. 313, no. 5783, pp. 72–74, July 2006. [Online]. Available: http://www.sciencemag.org/content/313/5783/72

[16] ——, "Generalizing Musical Intervals," *Journal of Music Theory*, vol. 53, no. 2, pp. 227–254, Sept. 2009. [Online]. Available: http://jmt.dukejournals.org/content/53/2/227

[17] ——, "Scale Theory, Serial Theory and Voice Leading," *Music Analysis*, vol. 27, no. 1, pp. 1–49, Mar. 2008. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1111/j.1468-2249.2008.00257.x/abstract

[18] G. Papadopoulos and G. Wiggins, "AI methods for algorithmic composition: A survey, a critical view and future prospects," in *AISB Symposium on Musical Creativity*. Edinburgh, UK, 1999, pp. 110–117.

[19] D. Cope, *Computer Models of Musical Creativity*. Cambridge, Mass: The MIT Press, Dec. 2005.

[20] R. Nikolaidis, B. Walker, and G. Weinberg, "Generative musical tension modeling and its application to dynamic sonification," *Computer Music Journal*, vol. 36, no. 1, pp. 55–64, 2012. [Online]. Available: http://www.mitpressjournals.org/doi/pdf/10.1162/COMJ_a_00105

[21] B. N. Walker and J. T. Cothran, "Sonification Sandbox: A graphical toolkit for auditory graphs," 2003. [Online]. Available: https://smartech.gatech.edu/handle/1853/50490

[22] S. Pauletto and A. Hunt, "A Toolkit for Interactive Sonification." in *ICAD*, 2004. [Online]. Available: http://www.icad.org/websiteV2.0/Conferences/ICAD2004/papers/pauletto_hunt.pdf

[23] O. Ben-Tal, J. Berger, B. Cook, M. Daniels, and G. Scavone, "Sonart: The sonification application research toolbox," July 2002. [Online]. Available: https://smartech.gatech.edu/handle/1853/51376

[24] J. Menezes, "SonData - Um toolkit para Sonorizao de Dados Interactiva," 2012. [Online]. Available: http://repositorio-aberto.up.pt/handle/10216/65260

[25] M. Bostock and J. Heer, "Protovis: A graphical toolkit for visualization," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 15, no. 6, pp. 1121–1128, 2009. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5290720

[26] M. Bostock, V. Ogievetsky, and J. Heer, "D$ˆ3$ data-driven documents," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 17, no. 12, pp. 2301–2309, 2011. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6064996

[27] N. Collins, A. McLean, J. Rohrhuber, and A. Ward, "Live coding in laptop performance," *Organised Sound*, vol. 8, no. 03, pp. 321–330, 2003. [Online]. Available: http://journals.cambridge.org/abstract_S135577180300030X

[28] J. Rohrhuber, A. de Campo, and R. Wieser, "Algorithms today - Notes on language design for just in time programming," *context*, vol. 1, p. 291, 2005. [Online]. Available: http://web.cecs.pdx.edu/~dreeder/site/nysc/doc/rohrhuber,etal--jit.pdf

[29] G. Essl and A. Mller, "Designing mobile musical instruments and environments with urmus," in *New Interfaces for Musical Expression*, 2010, pp. 76–81. [Online]. Available: http://web.eecs.umich.edu/~gessl/georg_papers/NIME10-UrMus.pdf

[30] C. Gerard and M. Sheller, *Salsa!: the rhythm of Latin music*. White Cliffs Media Co, 1989.