`dataSonification` Open Source Project for Real-Time Data Sonification

Edward P. Childs

dataSonification,
24 Old South Rd.,
Aquinnah, MA 02535 USA
maestrovt@gmail.com

John Stephens

1110 Turnpike Rd.
Norwich, VT 05055
john_stephens@alum.mit.edu

Benjamin Childs

6849 26th Ave NE
Seattle, WA 98115
ben@bchilds.com

ABSTRACT

A software platform for real-time data sonification is described in detail. The platform is designed primarily to process multiple real-time data streams simultaneously. The system was originally designed for processing financial data with the general goal to be able to monitor up to 20 different securities (stocks, bonds, financial indices) as their values were changing during a trading session. The sonifications were designed to make it easy to distinguish between different securities, and to convey as much information about each security's activity using the shortest possible sound duration. The software platform was designed using multiple threads with a gatekeeper function to manage simultaneous sonifications events without confusion or system failure. This paper announces the release of this software together with its financial data stream implementation to the open source community. It is hoped that sonification researchers particularly interested in real-time data will use and adapt the software to their needs.

1. INTRODUCTION

Sonification can be used effectively to monitor real-time data. As technology advances, more and more real-time data streams are generated in fields as diverse as intelligence, industry, medical, transport, financial, security, etc. While sophisticated visual displays and analysis algorithms can be used to process these data streams, the sheer time density of the data can overwhelm the monitoring systems and significant actionable events may be missed. Sonification can add the auditory faculty to existing data monitoring tools.

It can be argued that the auditory sense is superior in certain monitoring situations. For example, if a baseline state exists in a sonic format, and the listener has assimilated the sound of that baseline state, changes are easily discerned, even if the subject is concentrating on some other task [1]. There are many everyday examples of baseline monitoring, for example, familiarity with the sounds of one's automobile, computer, home heating system, neighborhood, etc. and the attention readily attracted to a "funny noise". In these cases, the system in question produces its own sounds by virtue of its physical or natural composition.

Traders working in traditional exchanges, set up in large rooms with hubbub and shouted bids and asks could get a "sense of the market", and even particulars on market movements from the sounds only. In modern visual electronic trading environments, however, traditional market sounds are absent. The sonification strategies implemented in the `dataSonification` software have been described elsewhere [2] [3] [4] [5]. Lodhi, Stockman, et al. have recently performed studies using the financial data implementation of the `dataSonification` package [6].

`dataSonification` has a modular design and its application is not limited to financial data. The system is designed so that the timing of the sonification will match the timing of the event to be sonified as closely as possible. In the financial data implementation, the goal was to reduce the latency of the system to less than a second, except in cases where multiple events to be sonified occurred close to the same time, in which case a `Gatekeeper` class would prioritize the sonifications to occur sequentially with a configured pause time between each.

2. SOFTWARE DESIGN

The core of the sonification software is written in the Java language and makes full use of the class-based, object-oriented structure. The design seeks to abstract the basic functionality of real-time data sonification into basic tasks:

1. Receive data.
2. Analyze data.
3. Sonify data according to specified analysis criteria.
4. Training on sonification (optional).

	s_id	enabled	ticker	description	analyzer	arranger
1	1	1	DJIA	Dow Jones - Motif D	MovementAnalyzer	OneNoteArranger
2	7	1	MSFT	Koto - Size and Price	SizeAnalyzer	SizeArranger
3	8	1	CLK14.NYM	Crude Oil May 14 - I	MovementAnalyzer	TwoNoteArranger
4	9	1	HOK14.NYM	Heating Oil Jul 13 - E	MovementAnalyzer	Beet5Arranger
5	10	1	IBM	IBM - Beethoven 5	MovementAnalyzer	Beet5Arranger
6	11	1	JNJ	JNJ - Beethoven 5th	MovementAnalyzer	Beet5Arranger
7	12	1	MOT	MOT - Beethoven 5	MovementAnalyzer	Beet5Arranger
8	13	1	MMM	3M - Target Analyze	TargetAnalyzer	FourNoteArranger
9	14	1	^IXIC	Nasdaq - Movement	MovementAnalyzer	ThreeNoteArranger
10	15	1	^GSPC	S & P 500 - Moveme	MovementAnalyzer	ThreeNoteArranger

Figure 1 SQL Configuration Database

It is up to the developer to extend, inherit or implement classes and interfaces, as appropriate to receive data from a specific source, analyze the data in a particular way, choose a sonification arrangement and configure the musical instrument upon which the data sonification is to be played.

The software is setup in modular format:

1. Configuration.
2. Data format.
3. Data source.
4. Data analyzer.
5. Sonification arranger.
6. Trainer.
7. Musical instrument.

The details of each numbered item are described in the following sections. Generic information will be provided, along with details of the specific financial software implementation. The reader should bear in mind that other implementations may be added to the basic framework.

2.1. Configuration

Configuration of all specific aspects of the sonification software is accomplished by extending the abstract class `Config`. Two configuration classes have been implemented to read configuration information either from a file or from an SQL database. A screenshot of a typical configuration, as viewed in the SQLite Database Browser, is shown in Figure 1. The reader may refer to this figure as other components of the software are described.

2.2. Data Format

The format of the data to be received for sonification is implemented by extending the abstract class `MessageComponent`. A generic XML format is available in the current implementation.

2.3. Data Source

The source of the data to be received for sonification is implemented by extending the abstract class `DataSource`. A Socket Listener is available in the current implementation. The external data source must be configured to send data to that Listener in the correct format over an agreed port. An example of a fully implemented data source, an Excel spreadsheet with a sonification plug-in, is described in Section 4.

2.4. Data Analyzer and Sonification Arranger

Incoming data must be analyzed to see if it meets the configured criteria for a sonification event. Those criteria depend on the context in which the data is to be monitored and the needs of the end user. In the financial data implementation, the end users were day traders interested in the difference between the current price of a security and its opening price. Some traders also wished to set a target price at which they would take some action, such as buying or selling. The target sonification would alert them if one of their securities was approaching the target price. Other traders were interested in the size of the trade, or the difference between the bid and ask prices in the case of government bond markets.

Both the analyzer and arranger components are constructed from the Superclass `SonificationComponent`. The abstract class `Analyzer` extends `SonificationComponent`. Its job is to analyze an incoming data event to see if the data is to be sonified, returning true if “sonify” or false if “do not sonify”. Analyzers for specific situations are created by extending abstract class `Analyzer`.

The sonification arranger determines exactly what sounds/notes should be played when the corresponding analysis class returns true. Thus for every arranger class there must be a corresponding analysis class. It is possible to have more than one arrangement for a given analyzer. The abstract class `Arranger` is constructed by extending the Superclass `SonificationComponent`. Its job is to return the appropriate instructions (i.e. notes) to the designated musical instrument.

For example, the `MovementAnalyzer` class (which extends the `Analyzer` class) calculates the difference between a base field and a current field. In the context of financial trading this could be the opening price and the current price. This calculated difference or increment is monitored by the `MovementAnalyzer` class until there is a change in that increment that is considered significant (as configured by the

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	Date	Time																	
2	6/28/2013	12:42:04						2%	2%										
3								Up From	Down From	Sonify	Item					Significant		Size	Sound
4	Ticker	Open	High	Last	Size	Bid	Ask	Open	Open	Function	Description	Instrument	Motif	s_id	Increment	Movement	Size	Change	Example
5		o	h	l1	k3	b3	b2												
6	^IXIC	4080.305	4110.462	4095.52	N/A	N/A	N/A	4161.91	3998.70	♫^IXIC♫	Nasdaq Composite	Organ	Three Note	14	3	4			
7	^FTSE	6584.17	6627.25	6625.25	N/A	N/A	N/A	6715.85	6452.49	♫^FTSE♫	FTSE 100	ACB-C	Up/Down	19	8	5			
8	^GSPC	1861.73	1869.63	1864.85	N/A	N/A	N/A	1898.96	1824.50	♫^GSPC♫	S & P 500	ACB-B	Three Note	15	0	4			
9	MSFT	40	40.195	40.01	2	700	40.01	40.80	39.20	♫MSFT♫	Microsoft	Koto - MIDI	Size Price	7	0	$0.10	700	100	
10	UBS	20.29	20.3657	20.25	N/A	19.5000	21.0000	20.70	19.88	♫UBS♫	UBS AG	ACB-A	Beethoven 5	21	-1	$0.02			
11	MMM	137.3	138.25	137.73	N/A	136.3000	138.2200	140.05	134.55	♫MMM♫	3 M	Tuba	Four Note	13	2	$0.20			
12	CLK14.NYM	104.54	104.61	103.88	N/A	103.8800	103.8900	106.63	102.45	♫CLK14.NYM♫	Crude Oil May 14	Double Bass	Two Note	8	-13	$0.05			
13	HOK14.NYM	3.0127	3.0127	2.9989	N/A	2.9984	2.9989	3.07	2.95	♫HOK14.NYM♫	Home Heating Oil May 14	Trombone	Beethoven 5	9	0	$0.02			
14	NGK14.NYM	4.756	4.789	4.776	N/A	4.7750	4.7770	4.85	4.66	♫NGK14.NYM♫	Natural Gas May 14	Tubular Bell	Three Note	18	0	$0.02			

Figure 2 Sample Excel Spreadsheet Data Source

trader). This "significant move" could be sonified in several ways. Traders interested only in up or down movement could choose the `TwoNoteArranger` class, which produces two pitches. The second pitch higher than the first indicates an uptick and the converse indicates a downtick. The `ThreeNoteArranger` class inserts a third reference pitch at the beginning to indicate the relationship of the uptick or downtick pitches to the security opening price. The `FourNoteArranger` appends a fourth pitch to indicate that a predetermined trading target is being approached. The `Beet5Arranger` class generates a motive based on the opening of Beethoven's 5th symphony, with pitches analogous to the `TwoNoteArranger` class, to slightly more dramatic effect.

There are many non-financial data monitoring situations to which the `MovementAnalyzer` class could be applied, such as a change in patient statistics in a hospital ward, a change in mass flow rate at some location in a processing plant, an increase in internet traffic over a specified route, an increased wind speed, etc.

The output of the arranger classes is a `SonifiableMusicShape` class. This class extends the Java Music Specification Language (JMSL) `MusicShape` class [6]. The `SonifiableMusicShape` class also implements the `Sonifiable` interface. This interface is necessary to enable the `Gatekeeper` class to manage simultaneous sonification events.

2.5. Trainer

The `dataSonification` software provides for an optional training component, the purpose of which is to provide more information on a specific sonification when it happens (perhaps using speech). The trainer would be disabled as soon as the end user no longer needs it. The abstract class `Trainer` extends the Superclass `SonificationComponent`. In some cases, the trainer could be used in lieu of a music-based sonification.

2.6. Musical Instruments

All musical instruments in the `dataSonification` package are derived from the JMSL [7] [15] [16] `com.softsynth.jmsl.Instrument`. The interface `ConfigurableInstrument` extends `Instrument` and adds a `setConfig` method which must be defined in any instrument implementation. Two instruments have been implemented in the `com.dataSonification` package, a sample instrument and a MIDI instrument. The `SampleInstrument` class extends the `TransposingSampleSustainingInstrument` which is part of the `com.softsynth.jmsl.jsyn` package [8]. The MIDI instrument `JavaMidiInstrument` extends `MidiScoreInstrument` from the `com.softsynth.jmsl.score.midi` package [7] and implements `ConfigurableInstrument`.

The sample instrument contains the entire expected infrastructure to load samples, interpolate between them, etc. The MIDI instrument is configured to play an instrument from the soundbank-deluxe.gm sound bank [9].

A large range of high quality orchestral samples originally generated from the Vienna Symphonic Library [10] are available along with more innovative sounds from the Acoustic Branding Audio Consulting Group [11].

3. EVENT MANAGEMENT

When real-time data is to be sonified, particularly in mission-critical applications, multiple sonifiable events may occur simultaneously. When this happens, the events are added to a queue in the `Gatekeeper` class and played in succession with a configurable pause between each sonification.

The success of this scheme depends on the time required for each sonification to play, and the configuration of the "significant move" described in Section 2.4. In the financial data implementation, the sonifications were designed to be economical, conveying the most information in the least time. For example the `ThreeNoteArranger` class, when played on a specific instrument to be associated with a specific security, would produce a sonification lasting about a 1 second and convey something like "The Dow Jones Industrial Average (DJIA) just moved up 10 points and is now 90 points above today's opening price." The frequency of the sonification was determined by the "significant move", in this example set to 10. Subsequent moves in the DJIA would not be sonified unless the index moved above 100 points over open, or below 80 points over open.

Judicious setting of this "significant move" for each security to be sonified is critical. The best strategy is to set the "significant move" high enough so that a quiet market would produce relatively few sonifications with minutes between each. A busy market would produce frequent sonifications, but in a quantity that could still be queued and played by the `Gatekeeper` class in a comprehensible fashion.

4. SPREADSHEET DATA SOURCE IMPLEMENTATION

The `dataSonification` package is setup to read a real-time data stream from any source over a socket, provided that the port number and data format are correctly configured. Included in the open source release is an Excel spreadsheet add-in, written in C#. It is assumed that real-time data from some other source is being imported into Excel, typically via another add-in. Many financial data providers include an Excel add-in [12]. There are other free add-ins which are designed to extract data from a wide range of financial data web sites [13]. The general methodology in [13] should however be applicable to any other type of web site which provides data being updated in real-time (e.g. weather data).

The C# Excel add-in provided with the package provides a "Sonify" function that can be called from within the spreadsheet. A sample spreadsheet using financial data is shown as Figure 2. The text entries in column A with the heading "Ticker" correspond to Yahoo ticker names. See column K "Item Description" for the security name. The numbers in columns B – G are imported from the Yahoo Finance website using the smf_addin [13], and are labeled accordingly. "Last" means the current price. Not all data items are available for all securities, as indicated by "N/A". Columns H and I contain sample calculations of a target price that might be set by a financial trader. If a given security reaches or approaches either target (in this sheet 2% up or 2% down from the opening price), the trader may wish to trigger a sonification, using the `FourNoteArranger` as described in Section 2.4. Column J is where the Sonify functions are called. The function is invoked in the standard Excel fashion, e.g. =Sonify(A6,D6,B6) in the case of the ^IXIC ticker. The first argument of the Sonify function is always the name of the data item, which also corresponds to the "ticker" field in SQL screen shot shown as Figure 1. The remaining arguments are numerical and depend on the specific data analysis to be performed. Column L describes the instrument chosen for the ticker and Column M identifies the sonification arrangement chosen. Columns N – R are configuration parameters specific to each ticker. Finally, Column S contains links to sound samples to be used by someone just learning the sonifications.

5. AVAILABILITY OF SOFTWARE

The software for this system, including the source code and documentation in both Java and C# may be downloaded from www.datasonification.com. Documentation and instructional videos are also available.

6. ACKNOWLEDGMENT

E.P.C thanks Ben Childs and John Stephens for their many hours of work on the concept and execution of this software system. The contribution of Kimo Johnson [14] to the original Java class structure is also gratefully acknowledged. Nick Didkovsky [7] and Phil Burk [8] provided extensive guidance on the integration of their respective Java music and sound libraries into `dataSonification`. Thanks also to Wilbert Hirsch and Patrick Langeslag [11] for some high quality sound samples and excellent suggestions for enhanced sonification schemes.

7. REFERENCES

[1] R. Näätänen, *Attention and Brain Function*. Hillsdale: Lawrence Erlbaum Associates, 1992

[2] P. Janata and E. Childs, "Marketbuzz: Sonification of Real-Time Financial Data," *Proc. 10th Int. Conf. on Auditory Display*, Sydney, Australia, July 6-9, 2004

[3] E. Childs, "Auditory Graphs of Real-Time Data," Proc. 11th Int. Conf. on Auditory Display, Limerick, Ireland, July 6-9, 2005

[4] Childs, Jr., E. P. et al., "System and Method for Musical Sonification of Data," *US Patent 7,138,575 B2*, Nov. 21, 2006

[5] Childs, E. P. et al., "System and Method for Musical Sonification of Data Parameters in a Data Stream," US Patent 7,135,635 B2, Nov. 14, 2006
[6] Lodhi, A. "Sonification of Stock Market Data", dissertation submitted in partial fulfillment of the requirements for MSc Software Engineering, Interactional Sound and Music (ISAM) Group, Queen Mary College, University of London, 2013
[7] http://www.algomusic.com, (Accessed 5/3/2014)
[8] http://www.softsynth.com, (Accessed 5/3/2014)
[9] http://www.oracle.com/technetwork/java/soundbanks-135798.html, (Accessed 5/3/2014)
[10] http://vsl.co.at/, (Accessed 5/3/2014)
[11] http://www.acoustic-branding.com/en/,(Accessed 5/3/2014)
[12] http://www.bloomberg.com/markets/, (Accessed 5/3/2014)
[13] https://groups.yahoo.com/neo/groups/smf_addin/info, (Accessed 5/3/2014)
[14] http://www.gelsight.com/, (Accessed 5/3/2014)
[15] N. Didkovsky and P.L. Burk, "Java Music Specification Language, an introduction and overview", *Proc. Int. Comp. Mus. Conf.*, 2001, pp. 123-126
[16] N. Didkovsky, "Java Music Specification Language, v103 update," *Proc. Int. Comp. Mus. Conf.*, 2004