TrioSon: A Graphical User Interface for Pattern Sonification

Charlie Cullen

Digital Media Centre

Dublin Institute of Technology (DIT)
Dublin,
Ireland
charlie.cullen@dmc.dit.ie

Eugene Coyle

Department of Control Systems and Electrical Engineering
Dublin Institute of Technology (DIT)
Dublin,
Ireland
eugene.coyle@dit.ie

ABSTRACT

The TrioSon software allows users to map musical patterns to input data variables via a graphical user interface (GUI). The application is a Java routine designed to take input files of standard Comma Separated Values (CSV) format and output Standard Midi Files (SMF) using the internal Java Sound API. TrioSon renders output Sonifications from input data files for up to 3 user-defined parameters, allocated as bass, chord and melody instruments for the purposes of arrangement. In this manner each parameter concerned is distinguished by its individual instrumental timbre, with the option of rendering any combination of 1 to 3 parameters as required.

The software parses indexed input data relating to individual variables for each user-defined parameter, and provides the means to allocate musical patterns to each variable for Sonification using drag and drop functionality. Control over the Rhythmic Parsing of the Sonification is provided, alongside individual control of the volume, panning, muting and timbre of each instrument in the trio. Sonifications can be rendered as full output files of the entire data, or can also be auditioned by index as required. This feature is designed to allow the user complete control over the data they are sonifying- either on an individual or collective basis.

Context for each output Sonification is provided by Midi events defined by the index of the input data, which are mapped to percussive timbres in the final SMF (via track 10). Java development provides the added advantage of portability, with the final application being small enough (200kb) to attach in an email document. It is hoped that the compact and intuitive nature of the application will make it a straightforward means of investigating the Sonification of data sets.

1. INTRODUCTION

Sonification can be defined as the use of non-speech audio to convey information [1]. The rapid increase in the amounts of data that many fields of modern research and endeavour have to process has led to the need for better tools and means of representation, and to this end audio representation is a viable and often powerful means of analysing data. Sound can convey significant amounts of information [2], and thus can be of great benefit in the analysis and understanding of data sets. Many modern electronic devices contain some means of generating an audio alert or warning (to signal a condition that requires some form of user interaction or intervention), ranging from car alarms through to the now familiar system sounds generated by computer hardware.

The development of such systems has occurred largely in ignorance of the overall benefit the Sonification has provided, and as such serves to suggest the potential of such information delivery techniques if properly utilised.

Pattern matching [3] is an area of data analysis that relies heavily on computational methods to distinguish possible trends or behaviours within a sequence or data set. It is suggested that as human intelligence and perception are designed specifically for the purpose of pattern matching, it may be of use to convey the particular data concerned in a format that lends itself towards human pattern recognition. Visual techniques for conveying data are well known (in anything from a line graph to fractal modelling), but Sonification for analysis is still relatively undervalued as a means of representing data for human recognition and understanding. The advantages of such Sonifications include the ability to deliver information to individuals or groups without any specific interaction on their part (such as an alarm sound in a public place), and also the relatively light processing overhead in conveying and storing such information (many Sonification techniques use the midi protocol to convey data and so do not require large computation or storage space). The capability of the human brain to process audio on a heirachal level (and quickly extract elements of information) suggests it to be a far more powerful means of communication and analysis than its present utilisation would often suggest.

Applications have been developed to facilitate data Sonification, and there are many good examples of task specific applications such as Caitlin [4] (for debugging computer software), MarketBuzz [5] (for stock market analysis) and Protein Music [6] (for analysing DNA sequences). More general data Sonification tools such as the Audio Abacus [7] or Sonification Sandbox [8] suggest great potential for the analysis of data in general, in an effort to ascertain the best means by which Sonification itself can be performed (before considering the data it will be applied to). It is suggested that an effective framework for data Sonification must first be developed in isolation, before it can be applied to any specific data with any real efficiency. The aim of the TrioSon software is to provide a means of considering the potential of pattern matching in data Sonification at a low level, with a view to eventually developing an effective method of pattern based data Sonification.

2. DESIGN AND IMPLEMENTATION

The TrioSon software is designed to allow pattern based Sonification of input data sets to be performed within a Graphical User Interface. By providing a simple and intuitive means of sonifying data, the software aims to allow the user to quickly assess input data as required. TrioSon was developed

using the Java language, which was deemed to be the best choice for a desktop application. Java applications are usually small, single file (JAR archive) releases which only require the presence of the Java Runtime Environment (JRE) in order to function. This compact construction differs significantly from applications developed using Visual Basic or C++, where an application can require many separate files to be installed to different locations on the target machine. Java is also a platform independent language that can be run efficiently on Windows, MacOS and Linux operating systems. This portability (alongside its small size) is one of the reasons for the proliferation of Java and JavaScript in applications for Internet Browsers and Mobile Devices. This portability again separates Java from languages such as C++ and Visual Basic, which often require extensive redevelopment in order to be compatible with different operating systems. Another important consideration in the application design was its potential migration to mobile devices such as cellphones and Personal Digital Assistants (PDA). It was intended that by developing the application in Java, the potential would exist to port the code onto such devices at a later stage.

2.1. Application design- MVC design and model

The first consideration in the application was a completely Object Oriented design. Java allows for OOP [9] (Object Oriented Programming) by way of the class based structure of the language, and this provided a means of segregating various operations within the code for separate consideration.
As part of the design of the TrioSon application, the Model-View-Controller [10] (MVC) architecture is used. MVC allows the GUI and the back end processing in the application to be separated for ease of design and implementation (Figure 1).

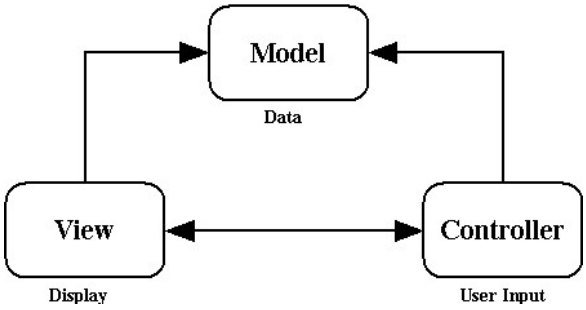

Figure 1. *Model-View-Controller Architecture*

In an MVC design, a Data Model class is constructed to hold all data and variables used by the application. The View classes relate to the GUI and its operation, with the Controller class defining all information input by the user. In Java, any class dealing with the input or display of information is hooked up to the Data Model using an Adapter [11], which effectively allows component classes in each View class to act as Controllers. Java defines the Listener interface as a means of classes obtaining information, and the Adapters allow a modified Model-View architecture to be created (with the Adapters acting as Listeners for the Data Model).
The Data Model could thus be designed separately from the GUI, allowing all the relevant requirements of the code to be specified as required. The application was required to have provision for bass and melody patterns, as well as chord intervals. In considering the Java Midi classes contained within the Java Sound API [12] (Application Programmer Interface), it was found that the Midi output functions were designed to take data of standard Midi format (such as Note Numbers and Message Codes) without additional conversion. Thus, the Data Model can be filled with the relevant patterns and chord intervals in a numerical format (Figure 2) for storage and rendering.

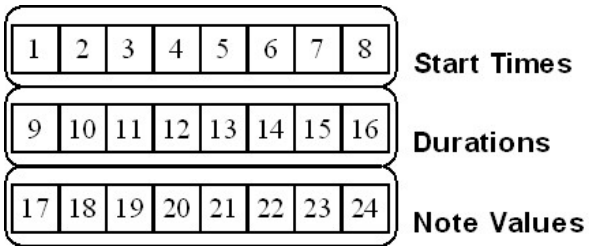

Figure 2. *Sample Melody Pattern Array in TrioSon Data Model*

The Rhythmic Parsing templates used in this software are also set up in this manner, and other variables that have known values or ranges (such as Instrumentation and Panning) are similarly defined as part of the Data Model template (Figure 3).

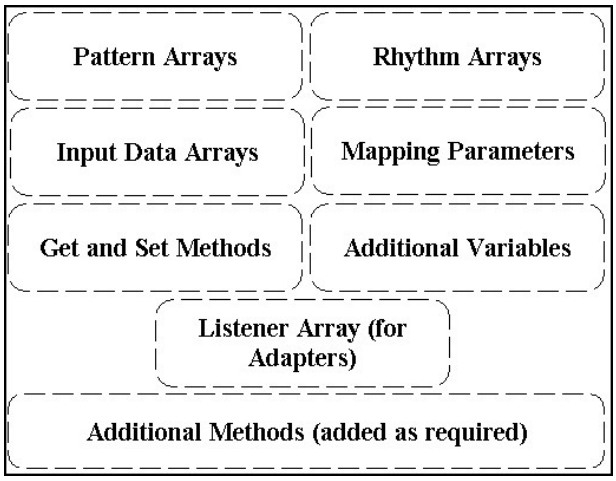

Figure 3. *TrioSon Data Model Template*

In the Java MV architecture, information is passed in and out of the Data Model (via the Adapters) using standard get and set methods, allowing template functions to be set up before any of the GUI functionality had been finalised. Consequently, the Data Model is organised into a template which can be amended as required, without any loss in functionality or change in implementation.

2.2. Input file format

With the basic framework for the application data in place, the next area of focus is the input of that data in the relevant format. The CSV format [13] (Figure 4) was chosen due to its inherent simplicity, a standard file contains one row of header information followed by data on every subsequent row.

Index	Name	Salary
1	Charlie	e10
2	John	e9
3	Clara	e15
4	Brian	e12
5	Marco	e10
6	Andreas	e11
7	Keith	e10

DataBase Format

```
Index,Name,Salary
1,Charlie,e10
2,John,e9
3,Clara,e15
4,Brian,e12
5,Marco,e10
6,Andreas,e11
7,Keith,e10
```

CSV File Format

Figure 4. *Example CSV file and its associated database representation*

Using this file type, any spreadsheet data can be used as required. The CSV format is a popular choice in simple spreadsheet and database applications (largely due to its simplicity), and so is considered the ideal file format for Sonification purposes. Once the CSV file format had been specified for input data, several functions were required to parse the information into the required format for the data model. A function was written to separate the header information from the following data, with the header being organised relative to the index of each data row in the file.

The file index can be regarded as the delimiter (Figure 5) which defines the separation of each set of values over time, allowing each set of Sonification patterns to represent a specific set of data values. Thus, the subsequent Rhythmic parsing of the Sonification can be used to separate each set of values by index as required by the user. This method of allocation requires that each index (i.e. each set of values) in the data set be listed under its own column heading, a heading that could be listed for assignment in the GUI.

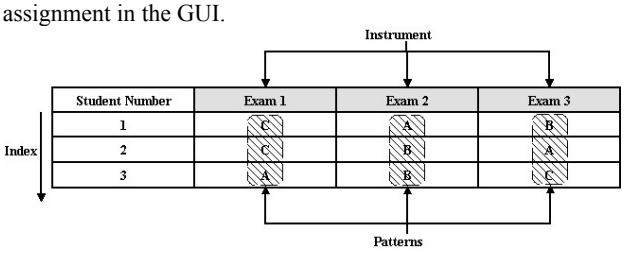

Figure 5. *Example Mapping scheme for Sonification of Student Results Dataset*

A second function is used to write the data from the input file into an array built by that function. In this manner, all data from the file is made available to the user by its header. Each individual parameter in the data could then be accessed from the data model when (or if) required. With the data input into the model, the application can now be manipulated by the user to produce an output Sonification of that data.

2.3. Data Parameter Allocation

The TrioSon GUI (Figure 6) is designed to be as straightforward and intuitive as possible. It is intended to have the minimum number of operations between file input and output, to allow the user quick and easy access to the Sonifications they have created.

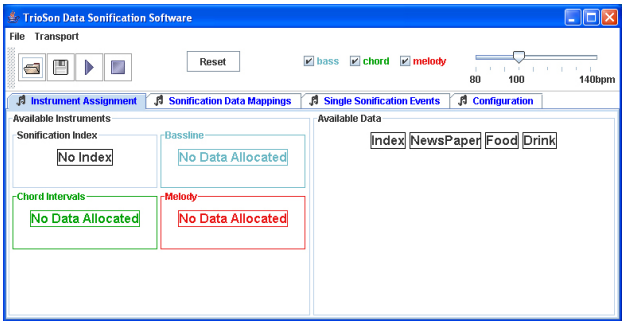

Figure 6. *TrioSon GUI front screen*

Each section of the GUI pertaining to a particular instrument is colour coded [14] (blue for bass, green for chords and red for melody), to help the user regard each instrument as an individual entity within the overall Sonification. The choice of colours was dictated by their proximity to one another within the visible spectrum- lower frequency blue through green to the higher frequency red.

The connotations of primary colours such as red and blue aims to segregate in simple terms, with the choice of green for chord relating to the frequency range occupied by chords in the framework (between low and high). This aspect of GUI functionality is intended to make the user as comfortable with the separation of each parameter as possible, while still retaining the overall cohesion of the trio arrangement.

All parameters in the input data file are displayed by their header, and the user can then drag and drop each header onto an instrument as desired. The index of the Sonification can also be assigned as required, allowing the user to organise a Sonification from a choice of more than one delimiter (if the input data has been so organised). Once each instrument has been assigned a data parameter, each unique variable for that parameter is then made available to the user for pattern allocation.

2.4. Pattern allocation

Each Instrument is given its own pattern allocation screen (Figure 7) that allows the user to drag and drop musical patterns onto targets representing each unique variable for that specified parameter.

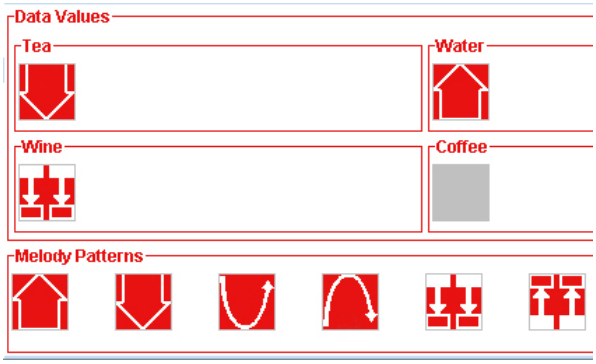

Figure 7. *TrioSon pattern allocation screen*

The available patterns are designed using Contour Icons [15] to allow the user to make high level choices about the patterns representing each data variable. By using Contour Icon patterns, the user can detect a pattern in an output Sonification by its melodic contour. This facility has been found to be a more

efficient means of pattern matching than by other non-graphical methods, while also providing the user with as straightforward an interface as possible.

2.5. Multi-threaded operation in Java

The Java programming standard allows for each Midi and GUI event to be performed in a separate thread [16] during operation. By moving playback and drag and drop operations into separate threads for execution (Figure 8), it is possible for the user to click and drag an icon while its relevant pattern (or chord) is played.

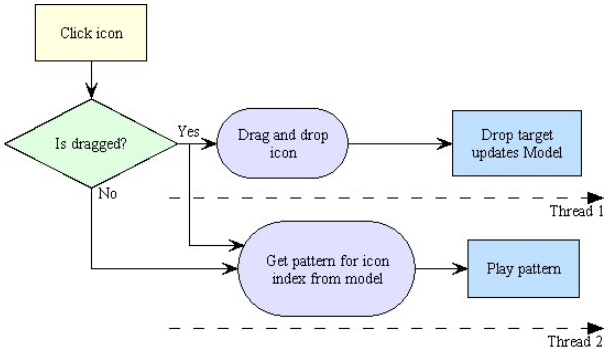

Figure 8. *Multi-threaded GUI operation*

This is an essential function within the GUI, allowing users to preview each pattern with a single click. It also provides an efficient means of repeating patterns to allow the user to become as familiar with their choices as possible.
By retaining a patterns signature during allocation to a specific variable a more intuitive link between pattern and variable can be achieved. By using multiple threads it is also possible to write a Midi file to disk while playing back the same file using the JavaMidi API, an element of application functionality that greatly reduced time delays during software operation.

2.6. Index, combination and rhythm screen

In order to provide the use with as much information as possible about the patterns they were required to detect, a Pattern Combination screen is provided (Figure 9).

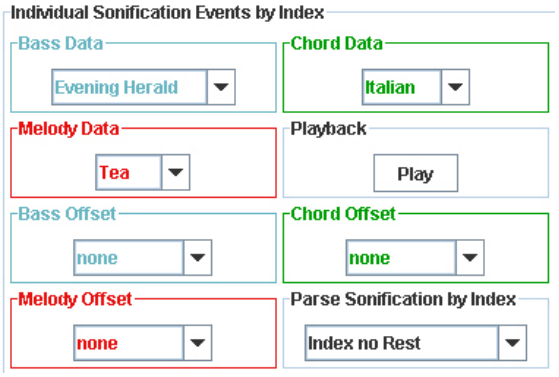

Figure 9. *TrioSon Single Events and Combinations screen*

The Java Combo boxes in the Pattern Combinations section allows each individual bass, chord and melody mapping to be played by the user. This feature was added to provide the user with the means to learn specific combinations before they listened to the full Sonification. In the above example (Figure 9), a data set of the answers to simple survey questions has been loaded into the application. The data specifies how many people (of those questioned) prefer a certain newspaper, type of food or beverage from a list of choices. By selecting different combinations using the drop down menus, a user can preview a particular combination of interest (in this case the Evening Herald newspaper, Italian food and Tea). In this manner, a user can detect how many people in the survey group chose a particular combination by learning that combination prior to (or during) listening to the full Sonification.
The user can also compare each set of combinations by index (Figure 10), in the above case pertaining to the newspaper, food and drink preferences of the survey group.

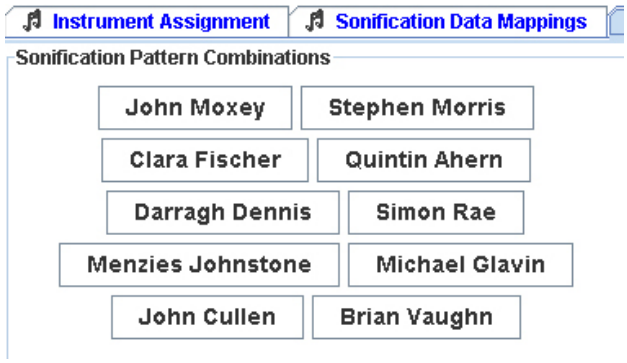

Figure 10. *Pattern combinations by index*

The survey group is listed by name (index), with each Java button playing the combination for that index. In this manner, a user could listen to the individual responses of a particular survey participant and define what their choices had been. A subjects set of choices can be compared with the pattern combination choices, to provide a straightforward method of pattern detection on an individual basis.
In the TrioSon software, provision is made for the Rhythmic Parsing [17] of the different combinations of events and offsets (Figure 9). A single event can be accompanied by up to 3 minim rests, or 2 events could be accompanied by 1 or 2 minim rests. In each case, the event itself is designed to last for no more than 1 minim. By providing the means to vary both the time between patterns and the time between each pattern in a set the user is able to structure the output Sonification to suit their own pace of comprehension [18] (Figure 11).

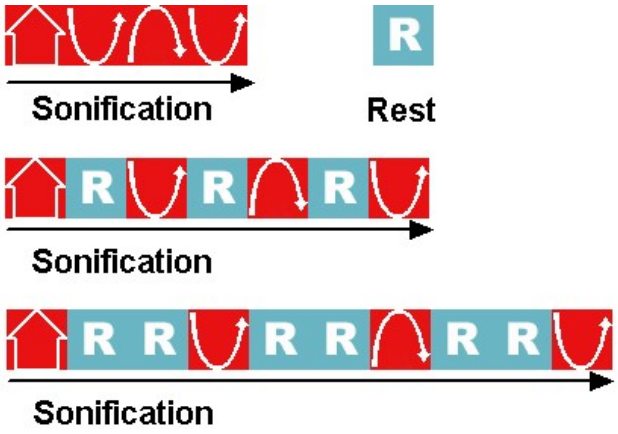

Figure 11. *Block diagram of the use of Rhythmic Parsing in a Sonification*

Placing short rhythmic gaps between events [19] in a Sonification allows the user time to detect and process the information in that Sonification more effectively than when no gap is present.
The offset between instruments (Figure 12) was also considered of great importance in multiple parameter Sonification, where it was desired to provide the user means of detecting each pattern for separate instruments in a sequence.

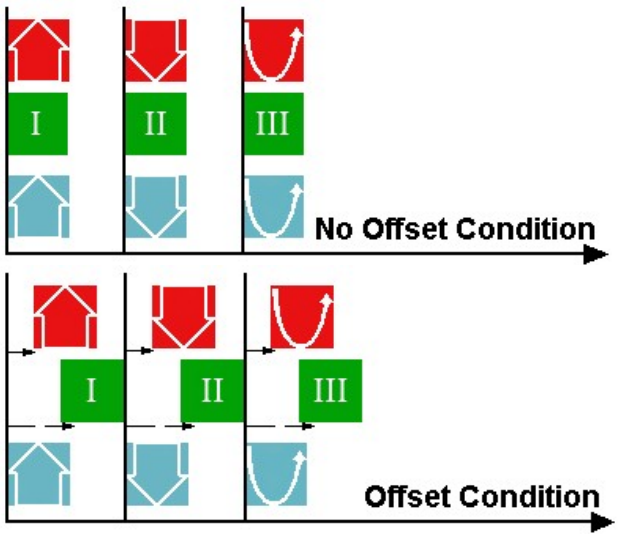

Figure 12. *Rhythmic offsets between parameters in a Sonification*

The use of offsets allows different patterns relating to the same index (such as a particular persons favourite food and drink) to be staggered relative to that index. In this manner, a user has the means to detect each pattern individually, while still retaining an overall segregation based on the Rhythmic Parsing of the Sonification.

2.7. Instrument, Volume and Panning Configuration

Each instrument used in the Sonification requires a General Midi patch to be assigned to it, in order that a useable Sonification could be produced. It is intended to include provision for external Midi devices (such as Samplers and Synthesiser Modules) in due course, but for initial purposes it is felt sufficient to provide General Midi timbres (Figure 13) via the onboard Soundcard of the relevant machine.

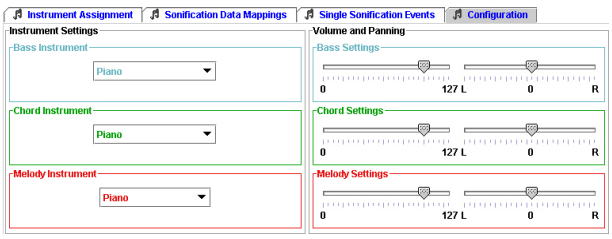

Figure 13. *TrioSon GUI Instrument Configuration Screen*

The individual volume and panning settings for each instrument are made available to the user. Spatialisation is considered an important part of the human hearing mechanism, and many experiments [20] on its effect have shown it to be a very important element in pattern recognition. The sliders provided allow the user to set the position of each instrument relative to others in the Sonification. This was felt to be an important consideration in multiple parameter Sonification, where the stereo position of each instrument would make it easier to detect than if centred monaurally. The guidelines specified for Earcon design [21] also suggest that volume is an important factor in inhibiting perception or masking tones. For this reason, it was felt important to allow the user to set relative volume levels for each instrument that they were comfortable with.

2.8. Output file format

A Midi [22] file contains sequence information that can be processed for output by any soundcard, keyboard or synthesiser module that has Midi capability. A file will typically contain a string of Midi messages, each of which define a certain operation to be performed by the device reading the file.
The Midi specification allows up to 16 Midi channels to be allocated (and controlled) at one time through a single Midi data stream. This means that any Midi file can potentially contain information pertaining to 16 different devices (or a single device with 16 different outputs). Midi messages are sent as binary (0 or 1) and defined as one Status byte (8 bits) followed by one or more Data bytes (Figure YY), with each message processed sequentially.

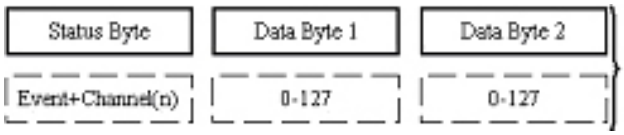

Figure 14. *Format of a Standard Midi Message*

Any Midi file written by software must define information in this manner, and choice of file format is also important. Format0 files contain Midi information for a single Midi track and therefore all note information in the file will be output on a single Midi channel. Format1 files can contain up to 16 separate tracks, each of which can hold Midi data that can then be sent to a required Midi channel. In the TrioSon application, the numerical note patterns held in the data model (see Figure 2) are written to a Midi file using the JavaSound API library routines, with the sequence of patterns being determined by the input data. Other information (such as instrumentation and configuration data) is written as Control Change messages, while timing information (such as Rhythmic Parsing and offset) is dictated by arrays held in the model that defines the time values of all output Midi messages.
Midi files are used by all aspects of the GUI output (such as individual and combination pattern playback), as it was found to be an efficient means of configuring and outputting data. The JavaSound API provides functions for real-time output of midi information, but this has been found to be unstable at times. Due to the small size of the midi files used, playing a temporary Midi file written for a specific pattern proved to be an effective means of allowing users to preview patterns during allocation.

2.9. Transport bar

In the TrioSon application, it was decided that the development of a transport section should be of prime importance. As a

functional element, the transport section was also given provision for tempo control and individual instrument muting (Figure 15).

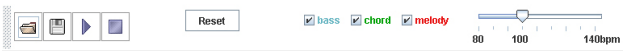

Figure 15. *TrioSon Transport Bar*

The flexibility of the Java Midi classes allows for the implementation of a tempo slider and checkboxes for each channel mute. The tempo slider was introduced to allow the user to listen to a Sonification at their own pace, which is particularly important during training and familiarisation. The channel mutes are included to allow single or dual parameter Sonification to take place as desired by the user. Although the ultimate aim of the application is to provide a multiple parameter Sonification framework, it is felt that lower parameter counts are essential for training- and indeed for many instances of single data listings. The entire transport section is embedded in a toolbar, which can be detached from the main GUI screen (by drag and drop) and relocated anywhere in the display area as required by the user. A Sonification can be played and stopped by the user, and the entire Sonification can also be written to file if desired. A Reset button was also added, to allow the user to recommence work with a new data file as required. The reset command re-initialises the data model and resets all parameters within the application GUI other than those related to instrument configuration. Instrument parameters (such as volume, panning, muting and patch selection) are left static as it was found that most users settle upon a preference of some kind after repeated use. It was felt that it would be more time consuming (and hence frustrating) to reset all parameters each time a new data file was loaded, rather than merely resetting those related to the data itself.

2.10. Context

Context [23] is provided by a single Midi click on ever quarter note in an output Sonification, with a different note being used for the first beat of every bar. Context is used to define each individual index in the output Sonification, and also provide the user with a means of synchronising to the patterns involved. Context is not implemented on individual patterns or combinations, as testing had shown that some users found it confusing during short sequences (of one pattern length). It is intended to investigate the potential of context more fully in future development.

3. CONCLUSIONS

The TrioSon software was developed to investigate various aspects of pattern matching and Sonification. The performance of listeners in multiple parameter conditions can be examined, as can the potential of different musical patterns. The use of Rhythmic Parsing can also be considered as a means of more effectively delivering information, and the compact nature of Java applications allows for testing to be performed under most conditions by a variety of users. This application has already been distributed to various users as part of ongoing testing, and initial results have shown it to be straightforward enough to be used with little training required. Sonification testing performed using the application showed it to be stable, and mo problems were encountered with the software during testing. A full release of the application (including source) is available via the DMC website (www.dmc.dit.ie) or by email (www.charlie.cullen@dmc.dit.ie). Development and amendment is encouraged by any interested parties, and it is hoped that the TrioSon application will eventually become a more comprehensive tool for data Sonification.

4. REFERENCES

[1] G Kramer(ed) et al, "Sonification Report: Status of the Field and Research Agenda," International Conference on Auditory Display (ICAD), 1997.

[2] HG Kaper, S Tipei, E Wiebel, "Data Sonification and Sound Visualization," Computing in Science and Engineering, vol. 1, no. 4, pp. 48-58, 1999.

[3] Web url: http://www.cs.ucr.edu/~stelo/pattern.html#Resources

[4] P Vickers, "Caitlin: Implementation of a Musical Program Auralisation System to Study the Effects of Debugging Tasks as performed by Novice Programmers," Doctoral Thesis, Loughborough University, 1999.

[5] P Janata, E Childs, "MarketBuzz: Sonification of Real-Time Financial Data," International Conference on Auditory Display (ICAD), 2004.

[6] RD King, CG Angus, "PM: Protein Music," Computer Applications in the Biosciences CABIOS, vol. 12, no. 3, pp. 251-252, 1996.

[7] BN Walker, J Lindsay, J Godfrey, "The Audio Abacus: Representing a Wide Range of Values with Accuracy and Precision," International Conference on Auditory Display (ICAD), 2004.

[8] BN Walker, J Cothran, "Sonification Sandbox: a Graphical Toolkit for Auditory Graphs," International Conference on Auditory Display (ICAD), 2003.

[9] J Hunt, The Hierarchical MVC, www.planetjava.com, Online Book.

[10] J Weber et al, Using Java, 2nd Edition, QUE 1996, ISBN: 0789709368.

[11] Swing Components Short Course, Sun MicroSystems, Online Tutorial.

[12] Java Sound API Documentation, Java™ 2 Platform Std.Ed.v1.4.2, Sun Microsystems.

[13] J Walkenbach, Microsoft® Excel 2000 Bible, Wiley, 1999, ISBN 0764532596.

[14] FA Wichmann, LT Sharpe, KR Gegenfurtner, "The Contributions of Color to Recognition Memory for Natural Scenes," Journal of Experimental Psychology: Learning, Memory, and Cognition, vol. 28, no. 3, pp. 509-520, 2002.

[15] C Cullen, E Coyle, "Musical Pattern Design Using Contour Icons," submitted to the 2005 International Conference on Auditory Display (ICAD) for review.

[16] C Austin, M Pawlan, "Writing Advanced Applications for the JavaTM Platform," Sun MicroSystems, Online Book.

[17] C Cullen, E Coyle, "Analysis of Data Sets Using Trio Sonification," Irish Signals and Systems Conference (ISSC), 2004.

[18] P Keller, "The Role of Metric Frameworks in The Processing and Representation of Musical Rhythm," Noetica Open Forum, Issue 8, 2004.

[19] S Katsuyuki et al, "Neural Representation of a Rhythm Depends on its Interval Ratio," The Journal of Neuroscience, vol. 19, no. 22, pp. 10074-81, 1999.

[20] D McGookin, SA Brewster, "Space, The Final Frontearcon: The Identification of Concurrently Presented Earcons in a Synthetic Spatialised Auditory Environment," International Conference on Auditory Display (ICAD), 2004.

[21] SA Brewster, PC Wright, DN Edwards, "Guidelines for the Creation of Earcons," British Human-Computer Interaction Group (BCS-HCI), vol. 2, pp. 155-159, 1995.

[22] P Messick, Maximum Midi: music applications in C++, Manning Publications 1998, ISBN 1-884777-44-9.

[23] BN Walker, DR Smith, "Tick-Marks, Axes and Labels: The Effect of Adding Context to Auditory Graphs," International Conference on Auditory Display (ICAD), 2004.