CAITLIN: A Musical Program Auralisation Tool to Assist Novice Programmers with Debugging

Paul Vickers
School of Computing and Mathematical Sciences
Liverpool John Moores University
Liverpool, UK
p.vickers@livjm.ac.uk

James L. Alty
LUTCHI Research Centre
Department of Computer Studies
Loughborough University
Loughborough, UK
j.l.alty@lboro.ac.uk

Abstract
In the field of auditory display relatively little work has focused on the use of sound to aid program debugging. This paper describes CAITLIN[1] a pre-processor for Turbo Pascal programs that musically auralises programs with a view to assisting novice programmers with locating errors in their code. A discussion follows of an experiment which showed that programmers could use the musical feedback to visualise and describe program structure. Conclusions and a discussion of future work are then given.

Keywords
Auralisation, audiolisation, auditory-display, musicode.

1. Introduction.
The term *software visualisation* suggests the idea of investigation using the visual sense alone. However, the aim of software visualisation is simply to improve the understanding of software [8]. Therefore, it makes sense to use sound if it possesses properties that facilitate software comprehension. Previous research shows that sound is a useful tool in the presentation of information to users. Examples include: Edwards' sound-enhanced word-processor for the blind [9], Gaver's *auditory icons* [11], Blattner's *earcons* [2], and Gaver's audio-enhanced graphical user interface [12].

Sonification, or the mapping of data to sound [20] demonstrates that large data sets can be represented using sound [3, 18, 21].

Audification [17], or the direct conversion of data to sound has been used to analyse large sets of seismic data that would be more difficult to visualise using graphics [13].

2. Program auralisation
Within the field of auditory-display research program *auralisation* is beginning to attract interest. Auralisation is the representation of program data (including execution state) using sound [15, 17]. The majority of efforts has been concerned with the auralisation of specific algorithms.

Brown and Hershberger [6] used music to enhance and complement an animation of a bubble sort algorithm. Other work has shown how sound can be used as the primary medium for visualising the state of parallel algorithms [10, 15].

The case for using music to aid debugging is supported by Francioni et al [14], although they felt that a visual presentation was also needed to provide a context or framework for the audio sound-track. Program state was captured as a set of trace data that were subsequently auralised and animated. Ways in which certain aspects of parallel programs can be auralised have been suggested by Jackson et al [15], but again the auralisation is of a 'post-mortem' nature.

Auralisation projects of note include the InfoSound system [22] by Sonnenwald et al, DiGiano et al's LogoMedia [7], Jameson's Sonnet system [16], Bock's Auditory Domain Specification Language (ADSL) [5] and Mathur's LISTEN system [4].

One stark omission from the existing literature is empirical evidence that program auralisation is actually useful. Preliminary experimentation showed that music can be used to visualise algorithm state [1]. In this experiment reaction to tasks that required the interpretation of musical output was gauged.

The first task showed that subjects were fairly accurate at estimating the difference (in semitones) between two musical pitches. The second task required subjects to sketch the perceived shape of short musical sequences. Subjects were generally able to pick up the basic shape being presented. As the experiment used musical sequences generated by a bubble sort the results indicate that music can aid visualisation of algorithm state.

3. The CAITLIN project
We now need some good experimentation to determine what is possible and practicable with program auralisation. The CAITLIN project aims to determine whether musical feedback can help novices to debug programs. Music might also be used to assist visually impaired programmers.

A pre-processor was constructed to auralise novices' Pascal programs. Experimentation will be undertaken to elicit evidence to support (we hope) the claim that musical feedback is useful in enhancing the debugging process.

Our auralisations are deliberately based on musical techniques. Music is an extremely powerful medium for the delivery of large amounts of data in parallel using techniques such as counterpoint and polyphony. Separate musical ideas can be delivered in parallel without confusion on the part of the listener if certain syntactic and semantic rules are followed. It makes sense to investigate whether music can be usefully employed in program auralisations. Other reasons for using music in program auralisations have been considered in earlier work [1] as have some of the arguments against using music [1]. One of the more enduring criticisms is that quantitative information cannot be conveyed by sound or music.

[1] http://www.cms.livjm.ac.uk/www/homepage/cmspvick/caitlin/caitlin.htm

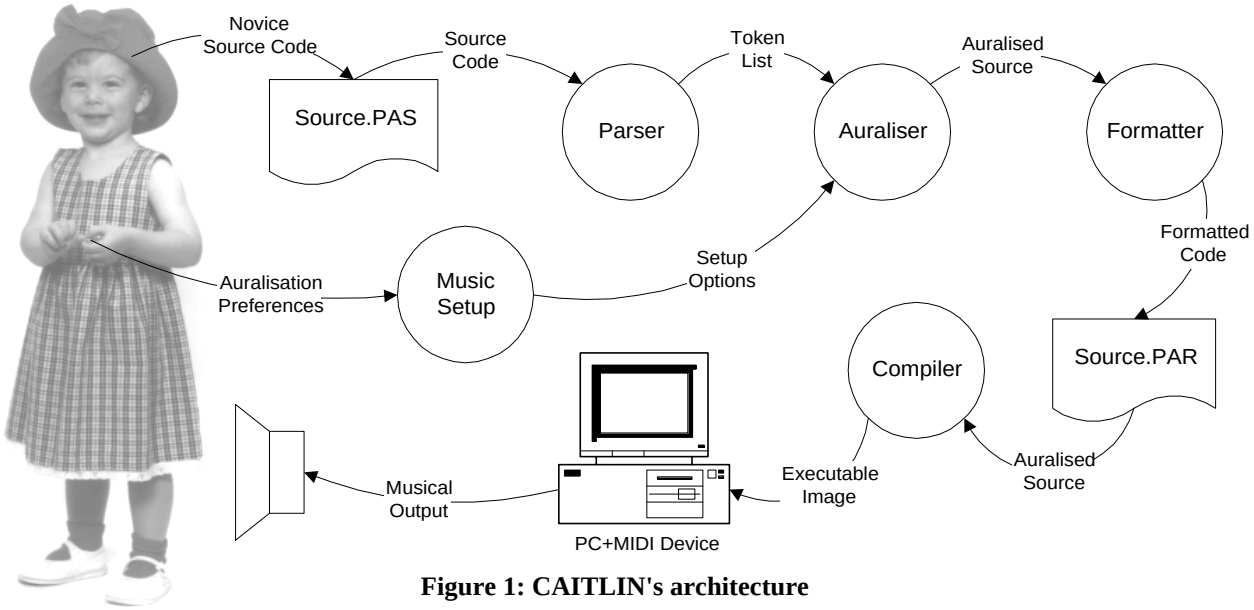

Figure 1: CAITLIN's architecture

The argument is that whilst most individuals can tell if a note increases or decreases in pitch, only trained musicians are able to determine exact intervals with any accuracy. However, quantitative information can be meaningfully described and presented in terms of its overall magnitude without needing to know its exact, discrete values. For instance, just as an unmarked thermometer allows visual judgement of the relative magnitudes of various temperatures, so musical pitch enables aural gauging of the relative values of different data.

Interestingly, nature already provides an acoustic thermometer in the form of the striped ground cricket. By listening to the cricket's chirps one can predict ambient temperature [19] (for example, twenty chirps per-second equates to a temperature of 88.6° Fahrenheit).

4. The CAITLIN pre-processor

The first stage of the project has been the construction of the CAITLIN system. CAITLIN is a non-invasive pre-processor that allows a novice programmer to auralise a program written in Turbo Pascal. Figure 1 shows the basic architecture of the system in terms of its functional units and their linkages.

Musical output is via MIDI to a multi-timbral synthesiser. Thus, the system can be implemented on a relatively modest platform comprising a personal computer and sound card with a General-MIDI-compatible instrument set.

CAITLIN is non-invasive; that is, it leaves the source program unchanged. The auralisations are effected by adding library routine calls to a copy of the program. The enhanced copy of the source program is compiled to produce an auralised executable image (see Figure 1). Because CAITLIN is designed to assist in debugging executable programs and not to help compile code, it will only accept a source program free of syntax errors.

On running CAITLIN (Figure 2) the user is presented with a screen similar in concept and layout to that of the Turbo Pascal Integrated Development Environment (IDE). A menu option allows the user to load a source program into memory which is then parsed and stored as tokens in memory. After loading the user can opt, via a menu (Figure 2), to auralise and then compile and run the auralised program or *musicode*.

Auralisation is done at the construct level. That is, a `WHILE` loop is auralised in one way and `REPEAT`, `FOR`, `CASE`, `IF...THEN...ELSE` and `WITH` constructs in others. The user may select, for each construct, the nature of the auralisation to be applied. Presently this is fairly simple-minded allowing selection of musical scale (e.g. major, minor etc.), default note length (e.g. eighth note), MIDI channel and instrument. The speed at which the music is heard is controlled by a user-definable tempo variable. All options can be saved to a configuration file.

For each construct the auralisation comprises three basic parts: a musical signature tune (leitmotif) to denote the commencement of the construct, a musical structure representing the execution of the construct and a signature to signal exit from the construct.

The contents of the musical structure within the construct will depend upon the construct's characteristics. Different constructs have different features which will be represented in various ways. To this end we have introduced the notion of the *point of interest* (POI). A point of interest is a feature of a construct, the details of which are of interest to the programmer during execution. For example, the `IF` construct has four POIs:

1. entry to the `IF` construct;
2. evaluation of the conditional expression;
3. execution of selected statement;
4. exit from the `IF` construct.

For each construct type the first and last POIs always denote entry to and exit from the construct respectively.

To enable to the listener to distinguish between the POI-1 of `FOR`, `WHILE` and `REPEAT` loops we have defined a short signature tune for each construct type. Thus, when a program is auralised the tune associated with `FOR` statements is inserted prior to each `FOR` loop and so on. A construct's last POI is auralised by playing a complement to its signature tune (such as playing it in reverse).

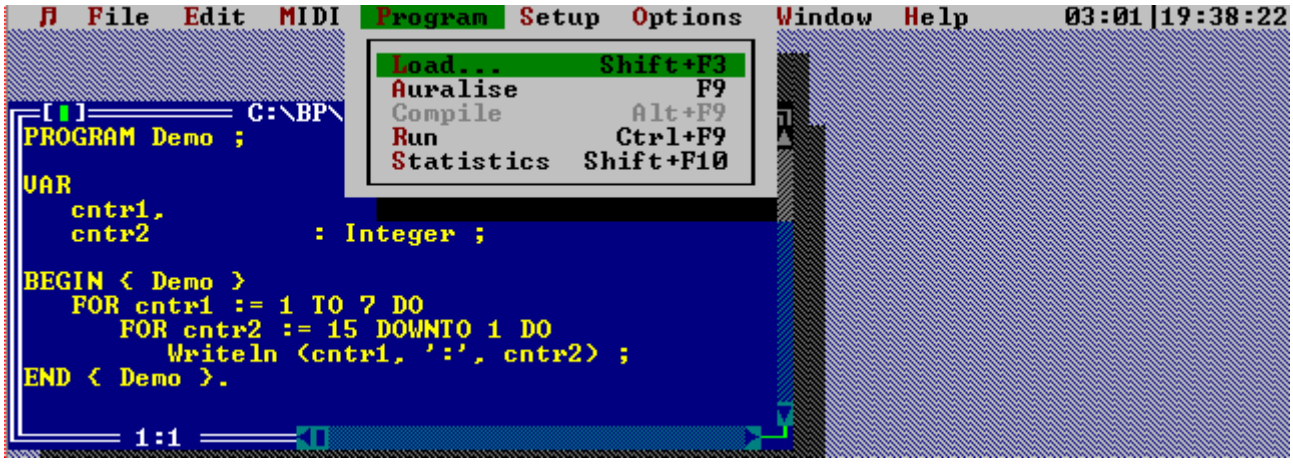

Figure 2: CAITLIN main screen

By defining a program in terms of its points of interest we build up an understanding of how each element and hence the whole program should sound. For example, we know that each `FOR` loop will be heard as a sequence of:

- playing of signature tune, followed by
- repetition of music denoting iterated statement execution, followed by
- playing of modified signature tune.

This is illustrated by Figure 2. The code window in Figure 2 shows a listing of a simple program employing two `FOR` loops. The auralisation employed in this example is straightforward. Code is inserted by CAITLIN so that each iteration of the outer loop generates a pitch of an ascending scale, the scale type being selected by the user. The inner loop plays a descending scale as the loop counter in this case is decremental. It must be stressed that the user only has access to the original source code and does not see the expanded auralised source.

5. Experimentation

Preliminary experimentation on CAITLIN was carried out. Subjects (eight faculty members) were first familiarised with the types of auralisation used by listening to ten examples[2]. Each example was accompanied by a narrative description of the program in question with source code available on request. Each sample auralisation could be repeated as many times as required. Following the familiarisation session subjects were presented with nine auralisations. For each auralisation they were asked to describe the structure of the program it represented. It should be noted that only audio cues were available; the output of the programs was not shown. Also, no facility was provided for changing any of the system parameters (such as instrument used etc.). The entire process took around 25 to 30 minutes.

The results suggest that, on the whole, the subjects were able to visualise program structure using only the auralisation (see Figure 3). Most subjects specified exactly the program structure represented by the auralisations. Where subjects did not give an exact description but could describe the essence of the structure, they were scored as 'nearly' correct (e.g., specifying a `FOR...TO` loop rather than a `FOR...DOWNTO` loop). Where the answer bore no correspondence to the actual then a score of 'no idea' was given. It is worth noting that the one subject who scored five 'no ideas' and four 'nearly' corrects claimed to lack familiarity with western music. More thorough experimentation should determine whether it was really this or simply a lack of familiarity with CAITLIN itself that was to blame.

Programs 8 and 9, which scored the fewest correct responses, contained combinations of `IF` and `IF...ELSE...IF` constructs. The poor scoring on these two examples is interesting because we have neglected (contrary to our previously stated guidelines) to provide an auralisation for the `IF` construct's final POI. Thus it is impossible to determine in all but a very few cases when an `IF` statement terminates. Program 8 involved nested `IF` statements. To deduce this required the listener to spot that the inner selection was played at an octave higher than the outer one; no example of nested selections was given in the familiarisation session. CAITLIN also fails to signal when an `ELSE` path exists for a selection whose conditional expression yields true.

6. Conclusions

In general, programmers understood what CAITLIN was doing and could follow the execution of simple programs. However, the ambiguity surrounding the `IF` statement shows that it is important to auralise exit from a construct as well as entry to it. The auralisation of the first and last POIs helped subjects to differentiate nested and sequential program structures. The background drone provided during execution of `WHILE` and `REPEAT` loops also assisted with this.

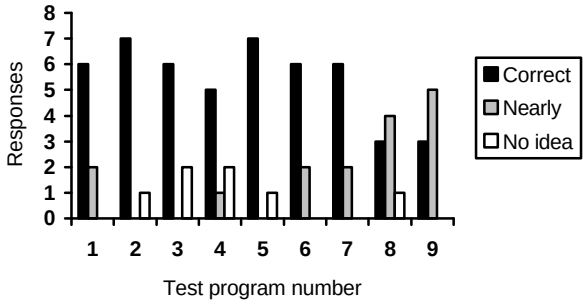

Figure 3: Preliminary experimental results

Instrument selection was seen to be very important. Subjects commented that it was easy to deconstruct auralisations in the mind when the timbres used for the various constructs were markedly different.

Careful attention must be paid to signature tune construction. One subject was unable to distinguish between the entry

[2] *http://www.cms.livjm.ac.uk/www/homepage/cmspvick/caitlin/tutorial.htm* provides access to these examples and the nine test auralisations used in the experiment.

and exit signatures of the FOR loop. Although this subject gave nine correct responses, more complex examples might cause confusion, especially when such loops are nested. The signature tune used for the REPEAT loop was more intricate than other signatures and did appear to confuse several subjects.

A proportion of the non-correct responses appear to be caused by subjects incorrectly remembering what each auralisation represented. One subject identified one auralisation as both a REPEAT loop and as a WHILE loop in consecutive test programs. He described his uncertainty as being caused by not remembering which tune was which. A longer familiarisation session may have improved his score.

The feedback from these preliminary experiments is being used to develop the next version of CAITLIN which will be used by novice programmers. Further experimentation will determine whether the novice programmer can use auralisations to help debug programs.

7. References

[1] Alty, J. L., Can We Use Music in Computer-Human Communication?, in *People and Computers X*, D. Diaper and R. Winder, Eds. Cambridge: Cambridge University Press, 1995.

[2] Blattner, M. M., Sumikawa, D. A. and Greenberg, R. M., Earcons and Icons: Their Structure and Common Design Principles, *Human Computer Interaction*, **4**, 1989, pp. 11-44.

[3] Bly, S. A., Communicating with Sound, in *Proc. CHI '82* (1982), New York: ACM Press/Addison-Wesley, pp. 371-375

[4] Boardman, D. B. and Mathur, A. P., *Preliminary Report on Design Rationale, Syntax, and Semantics of LSL: A Specification Language for Program Auralization*, W. Lafayette, IN: Dept. of Computer Sciences, Purdue University Sept. 21, 1993.

[5] Bock, D. S., Auditory Software Fault Diagnosis Using a Sound Domain Specification Language, Ph.D. thesis, Syracuse University, Syracuse, 1995.

[6] Brown, M. H. and Hershberger, J., Color and Sound in Algorithm Animation, *Computer*, **25** (12), 1992, pp. 52-63.

[7] DiGiano, C. J. and Baecker, R. M., Program Auralization: Sound Enhancements to the Programming Environment, in *Proc. Graphics Interface '92* (1992), pp. 44-52

[8] Domingue, J., Price, B. A. and Eisenstadt, M., A Framework for Describing and Implementing Software Visualization Systems, in *Proc. Graphics Interface* (1992), pp. 53

[9] Edwards, A. D. N., Soundtrack: An Auditory Interface for Blind Users, *Human Computer Interaction*, **4** (1), 1989, pp. 45-66.

[10] Francioni, J. M. and Rover, D. T., Visual-Aural Representations of Performance for a Scalable Application Program, in *Proc. High Performance Computing Conference* (1992), pp. 433-440

[11] Gaver, W. W., Auditory Icons: Using Sound in Computer Interfaces, *Human Computer Interaction*, **2**, 1986, pp. 167-177.

[12] Gaver, W. W., The SonicFinder: An Interface that Uses Auditory Icons, *Human Computer Interaction*, **4** (1), 1989, pp. 67-94.

[13] Hayward, C., Listening to the Earth Sing, in *Auditory Display*, vol. **XVIII**, *Santa Fe Institute, Studies in the Sciences of Complexity Proceedings*, G. Kramer, Ed. Reading, MA: Addison-Wesley, 1994, pp. 369-404.

[14] Jackson, J. A. and Francioni, J. M., Aural Signatures of Parallel Programs, in *Proc. Twenty-Fifth Hawaii International Conference on System Sciences* (1992), pp. 218-229

[15] Jackson, J. A. and Francioni, J. M., Synchronization of Visual and Aural Parallel Program Performance Data, in *Auditory Display*, vol. **XVIII**, *Santa Fe Institute, Studies in the Sciences of Complexity Proceedings*, G. Kramer, Ed. Reading, MA: Addison-Wesley, 1994, pp. 291-306.

[16] Jameson, D. H., Sonnet: Audio-Enhanced Monitoring and Debugging, in *Auditory Display*, vol. **XVIII**, *Santa Fe Institute, Studies in the Sciences of Complexity Proceedings*, G. Kramer, Ed. Reading, MA: Addison-Wesley, 1994, pp. 253-265.

[17] Kramer, G., Preface, in *Auditory Display*, vol. **XVIII**, *Santa Fe Institute, Studies in the Sciences of Complexity Proceedings*, G. Kramer, Ed. Reading, MA: Addison-Wesley, 1994, pp. xxiii-xxxviii.

[18] Mezrich, J. J., Frysinger, S. and Slivjanovski, R., Dynamic Representation of Multivariate Time Series Data, *Journal of the American Statistical Association*, **79** (385), 1984, pp. 34-40.

[19] Pierce, G. W., *The Songs of Insects* Harvard University Press, 1949.

[20] Scaletti, C., Sound Synthesis Algorithms for Auditory Data Representation, in *Auditory Display*, vol. **XVIII**, *Santa Fe Institute, Studies in the Sciences of Complexity Proceedings*, G. Kramer, Ed. Reading, MA: Addison-Wesley, 1994, pp. 223-252.

[21] Scaletti, C. and Craig, A. B., Using Sound to Extract Meaning from Complex Data, in *Extracting Meaning from Complex Data: Processing, Display, Interaction*, vol. **1259**, E. J. Farrel, Ed. San Jose, California: SPIE, 1990, pp. 207-219.

[22] Sonnenwald, D. H., Gopinath, B., Haberman, G. O., Keese, W. M., III and Myers, J. S., InfoSound: An Audio Aid to Program Comprehension, in *Proc. Twenty-Third Hawaii International Conference on System Sciences*, **11** (1990), IEEE Computer Society Press, pp. 541-546