CAITLIN: a musical program auralisation tool to assist novice programmers with debugging

This item was submitted to Loughborough University's Institutional Repository by the/an author.

Citation: VICKERS, P. and ALTY, J.L., 1996. CAITLIN: a musical program auralisation tool to assist novice programmers with debugging. IN: The Proceedings of the 3rd International Conference on Auditory Display, ICAD'96, Palo Alto, California, November 4-6, 1996

Additional Information:

- This is a conference paper

Metadata Record: [https://dspace.lboro.ac.uk/2134/3355](https://dspace.lboro.ac.uk/2134/3355)

Publisher: ICAD

Please cite the published version.
This item was submitted to Loughborough’s Institutional Repository by the author and is made available under the following Creative Commons Licence conditions.

For the full text of this licence, please go to:
http://creativecommons.org/licenses/by-nc-nd/2.5/
The CAITLIN Auralization System: Hierarchical Leitmotif Design as a Clue to Program Comprehension

Paul Vickers
ARCADIA Laboratory
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**

Early experiments have suggested that program auralization can convey information about program structure [5]. Languages like Pascal contain classes of construct that are similar in nature allowing hierarchical classification of their features. This taxonomy can be reflected in the design of musical signatures which are used within the CAITLIN program auralization system. Experiments using these hierarchical leitmotifs should (see note in EXPERIMENT section) indicate that their similarities can be put to good use in communicating information about program structure and state.

**KEYWORDS**

Auralization, visualization, music.

**INTRODUCTION**

The CAITLIN pre-processor\(^1\) [5] provides musical auralizations of Turbo\(^2\) Pascal programs. The system adds calls to library routines to a copy of a program source. These routines generate the musical auralizations which are played via MIDI on a Boss DS-330 multi-timbral synthesiser. The arguments for using musical techniques have been discussed more elsewhere [1, 2, 5], but in summary:

- Western musical structures, whether by independent evolution or by cultural imposition [4], are widely accepted across the world.
- The information contained in a large scale musical work approaches that of a moving video (a typical audio CD contains hundreds of megabytes). The potential exists for using music to successfully transmit complex information.
- Music forms a large part of peoples’ daily lives. It can be very memorable and durable. Most people are resonably familiar with the language of music in their own culture.
- Music involves the simultaneous transmission of a set of complex ideas related over time, within an established semantic framework. The job of a composer is to use musical resources and techniques to enable a listener to successfully disambiguate such information.
- Music may offer an important communication channel for blind or partially-sighted users.

Auralizations are currently carried out at the construct level. That is, only the major Pascal constructs (loops and selections) are assigned musical representations. The ultimate aim of the project is to construct a musical debugging environment to assist novice programmers. But before this can be done it must be determined whether music is a useful communication medium in program comprehension. For the purposes of the current work the focus has been deliberately constrained to include only the major constructs and structural features of the computer program. Even with this limitation a measure of success has already been achieved in using the auralizations to help locate bugs in short programs [2].

**POINTS OF INTEREST**

CAITLIN uses the notion of the point of interest (POI) in its auralizations. We defined the POI as “a feature of a construct, the details of which are of interest to the programmer during execution” [5]. Each POI is represented by a musical device or leitmotif. In music a leitmotif is a recurring theme associated with a particular thought or character. In the first CAITLIN prototype the leitmotifs, whilst musical in structure, were largely arbitrary in their design, the only consideration being to make each one distinct from the others to avoid ambiguity.

As a starting point this approach was successful, but was limited in that a program’s entire auralization did not have a unifying musical structure. Indeed, different constructs could be assigned to different musical scales. The effect was akin to using fragments of separate songs with different meters and musical keys as opposed to passages from a single musical piece linked by a common time signature, tempo and key. The purpose of the current work was to develop a more unified approach to leitmotif design to impose a more formal structure on the musical auralizations.

**PREVIOUS RESULTS**

A previous experiment [5] indicated that programmers could follow the execution of simple programs by listening to their auralizations. Furthermore, they were able to describe the structure of the programs from the information presented by the auralizations. However, a number of limitations in the implementation of the auralizations were highlighted which detracted from the overall effectiveness of the system. These included:

- failure to auralize all POIs of a construct which led to ambiguity;
- poorly designed leitmotifs which led to some failures in subjects’ identification of constructs;
- arbitrariness of leitmotif design led to poor association and recall in some subjects causing the same leitmotif to be identified as different constructs on different occasions.

**PROGRAM FEATURES AND REDESIGN OF LEITMOTIFS**

Having achieved a measure of success with simplistic musical devices we concentrated on the details of leitmotif design with a

---

\(^{1}\) http://www.cms.livjm.ac.uk/www/homepage/cmspvick/caitlin/caitlin.htm
view to making the auralizations more integrated and unified.

Programming languages offer the programmer a range of tools for achieving similar ends. Pascal provides three iteration constructs—WHILE, REPEAT and FOR. Each allows iterative execution of code but differs from the others in the way the looping is controlled. So, the three loop constructs are different but share certain characteristics. Similarity is also found in the selection statements. IF, IF...ELSE, and CASE provide selective execution of statements but use different mechanisms to accomplish it.

So, there is an inherent taxonomy of constructs in the language. The motivation behind the redesigned leitmotifs was to model this hierarchy musically. In this way there would be a theme denoting iteration and another theme to denote selection. Within each, variations of the theme would be used to represent the individual constructs. All selections would thus sound similar but entirely different from the loop constructs. Figure 1 shows how the selection and iteration constructs of Pascal are organised and how this structure can be modelled in a hierarchical leitmotif design.

The approach taken was to use a chord-based motif for the iterations and a melodic device for the selections. For the two classes of construct there are three fundamental points of interest that require auralization: entry to the construct, execution of the construct’s body and exit from the construct. The entry and exit POIs are obviously related as the pair serves to parenthesize the construct. Therefore, these two POIs are modelled by related leitmotifs where the exit motif provides closure for the entry theme. Figure 2 gives the generic selection theme. There is a rising scale signifying entry to the selection construct and a descending scale denoting exit. The underlying theme was reworked for each of the individual selection constructs. This was accomplished by changing the rhythmic patterns of the theme for each leitmotif.

**Figure 1—Taxonomy for leitmotif design**

The motif for the simple IF statement (i.e., an IF without an ELSE path) is given as Figure 3. We observe the same basic melodic theme but with a modified rhythm.

**Figure 2—General selection theme**

The general iteration is given as Figure 4, which shows a simple chord progression of tonic to tonic. (I-I) to denote entry to and exit from the loop construct.

**Figure 4—General iteration theme**

The FOR loop’s variation is shown in Figure 5. We see the overall progression retained but with some additional chords and rhythmical variations. More chords are added to represent the loop bodies (q.v.) leading to quite distinct chord progressions for each loop construct (although each begins and ends on the tonic).

**Figure 5—FOR statement theme**

**CONSTRUCT BODIES**

The principle of construct similarity was used in the design of motifs for the construct bodies. IF and IF...ELSE involve evaluation of boolean expressions. Iterations use the value of a boolean expression to control the repeated execution of a group of statements. We represented boolean evaluations by using a motif in a major key for true results and minor keys for false results. A potential danger lies in the tendency to associate music in major keys with happiness and the minor modes with sadness. We may subconsciously equate a boolean true (major) as being ‘good’ whilst seeing the false as bad (minor). But for the continued execution of the REPEAT loop, the boolean expression must be false.

The mapping of true to major and false to minor has been incorporated in the various points of interest of the constructs. Figure 6 shows the auralization of a simple IF statement whose conditional expression yields true.

**Figure 6—IF statement yielding 'true'**

Figure 7 shows the same statement but this time the expression yields false. Notice how the exit motif is also changed to a minor key to reinforce this. Similar devices are used in IF...ELSE and CASE.

**Figure 7—IF statement yielding 'False'**
As a loop implies the construct’s persistence over time a background drone is added to the iterations’ bodies to reinforce in the listener’s mind that everything that is happening is doing so within a loop. We need to know when the individual loop iterations occur and when the controlling boolean expression is evaluated. For the REPEAT and WHILE loops a simple major/minor chord device is used when the loop condition is tested; this will be heard immediately after entry to the WHILE loop, but after the iterated statements for the REPEAT loop. Each time one of these chord devices is heard we know that the loop has reached its decision point. The null WHILE loop (where the terminating condition is true upon entry) would thus be heard as a sequence of entry motif followed by minor chord motif for condition evaluation followed by the exit motif.

The FOR loop is count-controlled, a loop invariant taking incremental steps from a starting value to an end value. To denote this stepping up (or down) of the invariant, the pitch of the drone in the FOR loop is increased (or decreased for the FOR…DOWNTO) by one diatonic step with each repetition.

[Note: depending on space limitations in final papers, it may be possible to include musical scores for the complete loop structures here.]

VARIATIONS ON A THEME

This idea of variations on a theme ensures that all selections sound like each other but can be distinguished by their individual mutations of the class motif. The reason for doing this is not simply one of organizational convenience, although such categorization can be useful cognitively; applying these techniques provides us with a means of program comprehension at different levels of abstraction and also with a way of conveying spatial information temporally. The abstraction is achieved because one can choose to listen to a program’s auralization in terms of its overall structure (e.g., a selection followed by a loop etc.) or in terms of its details (e.g., an IF…ELSE followed by a WHILE). Further abstractions could be achieved by providing selective auralizations in terms of:

- classes of construct to auralize
- number of iterations of a loop
- nesting depth of constructs

AURAL PRESENTATION OF TEMPORAL AND SPATIAL INFORMATION

The categorization of constructs also enables a musical portrayal of spatial program features. One of the driving forces behind program auralization is that sound is a temporal medium and program execution is a temporal phenomenon; therefore, it makes sense to explore the possibilities of mapping the latter to the former. Through auralization we can listen to the execution of a program and make inferences about its state. But restricting an auralization to temporal detail alone may lead to a loss of quality.

Consider the code fragment in Figure 8. If all occurrences of IF statements sounded alike and if the value of ‘a’ were greater than 3 then it would be impossible to determine, from the auralization alone, whether one is hearing a simple selection (first IF in Figure 8) or one with an ELSE path but where the ELSE part was not followed (second IF). By categorizing the constructs and building this into the theme tunes we can avoid this ambiguity. CAITLIN uses a modified form of the simple IF statement motif (Figure 3) to represent the IF statement that has an ELSE part (see Figure 9).

![Figure 9—IF…ELSE motif](image)

An advantage of this approach is that when an IF…ELSE occurs and the initial condition is false the listener is not caught unawares by the subsequent occurrence of an ELSE structure in the auralization. By setting up this anticipation\(^2\) by the listener of possible future events CAITLIN creates a sort of construct ‘footprint’ which shows not just where we have been but also where we might go. Such an auralization is able to capture the spatial information relating to the presence of an ELSE path; the difference between the two constructs in Figure 8 is readily made apparent in the auralization.

The CASE statement (Figure 10) is another example of how CAITLIN conveys spatial information within the musical framework.

![Figure 10—CASE statement](image)

Like IF…ELSE…IF, CASE allows for alternative courses of action depending on a variable’s value. Of interest to the programmer is which instance of the CASE labels (if any) produces a match. Unlike the IF…ELSE…IF which carries out its comparisons of the various (nested) conditional expressions sequentially, no such ordering is implied by the CASE. However, it is convenient for us to think of the variable as matching the first, second, third etc. instance of the CASE labels. In Figure 10, if ‘x’ had the value 3, then we would say that the second label produced a match.

This is a spatial judgement because the second label is lower down the list than the first. This fact is communicated by signalling the presence of each label in turn (by a percussive sound). If a particular label produces a match then a major chord is also sounded. The resultant auralization gives the effect of the computer stepping over each label in turn until the end (or ELSE part) is reached or a match occurs.

Another spatial element that can be mapped to sound is construct nesting. Programmers show this visually by indenting

\(^2\) See Robert Jourdain’s ‘Music, the Brain and Ecstasy’ for an interesting description of musical anticipation [3].
the code for each level of nesting. Currently CAITLIN represents nesting depth by increasing the octave of nested components. However, it only takes five or six levels of nesting before the pitch becomes too high to be useful. Other possible mappings include position within the stereo field or using background drones for each construct. There are limitations with each approach and further research will be conducted into how best to map nesting to music.

EXPERIMENT
Note: as the experiment is in progress, results are not available at the time of writing this abstract. However, results will be included in the finished paper (assuming acceptance, of course).

To determine whether this hierarchical design approach is useable a small study is being conducted. Faculty members at John Moores University with experience in teaching programming are being used as the subjects. The aim is to determine whether having heard an example auralization of the iteration and selection classes the subjects could then assign other auralizations to their correct class type. A similar experiment to that previously described [5] will also be performed to see if the new leitmotif designs improve performance in describing a program’s structure from its auralization.

In the first test is is intended to explain to subjects the theory of the auralizations employed by CAITLIN. The notion of construct similarity is also to be explained. The general iteration and selection themes will then be presented to be followed a series of auralizations. For each subject the auralizations will be in random order and will be separated by a representation of the generic themes. Each auralization is of a single construct. For each, the subjects are asked to classify the construct as either a selection or an iteration and then to try to describe its specific nature (e.g., an iteration: WHILE loop). Examples of all constructs are given including cases of true and false boolean expressions.

The second test is a repeat of the earlier experiment but with the new motifs. Space does not allow a full description of the method (see the earlier paper [5] for this).

RESULTS
Results from the first test will be analysed to see if there is any support for the ideas set out in the paper. It is expected that subjects will tend to correctly classify the class of each auralization. We will also examine the results to see if a learning effect is evident over the duration of the experiment.

The second test is expected to show an improvement in accuracy of construct recognition using the new CAITLIN system. As ambiguities of the first system have been (we hope) removed, programs 8 and 9 should score much more favourably than before.

CONCLUSION
We hope to demonstrated that aspects of program structure can be mapped successfully to music. A further set of experiments will involve novice programmers. Investigation of the best way to present construct nesting needs to be undertaken.

REFERENCES

AUDIO EXAMPLES
It is proposed to play examples of the auralizations described as part of the paper presentation. We would intend to illustrate the various ideas raised in the paper with appropriate audio examples. As each auralization is short, it is feasible to play a number of auralizations within the half-hour presentation.