Program Auralization: Author's comments on Vickers and Alty, ICAD 2000

PAUL VICKERS
Northumbria University

In this paper, we reflect upon the investigations into external auditory representations of programs (*program auralization*) reported by Vickers and Alty at ICAD 2000. First, we place the work in its historical and thematic context and explore the motivation that lay behind it. We then outline the process by which we got to the stage of being able to report empirical results in 2000 and compare the work with that done by other researchers in the field. Finally, we assess the major contribution that this work made to the field of auditory display and look to the future outlining the work still to be done since the paper was first published (we also look at work done by others in this area since 2000).

Categories and Subject Descriptors: H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*Auditory (non-speech) feedback*; H.5.1 [**Information Interfaces and Presentation**]: Multimedia Information Systems—*Audio input/output*; H.5.5 [**Information Interfaces and Presentation**]: Sound and Music Computing—*Methodologies and techniques*

General Terms: Human factors, Languages

Additional Key Words and Phrases: Music, auralization, debugging, Pascal

1. HISTORICAL CONTEXT AND PROVENANCE

Program auralization is the mapping of events and objects in the program domain to the auditory domain. The results presented at ICAD were the outcome of the `CAITLIN` (see www.auralisation.org) research project into program auralization that James Alty and I started in late 1994. The motivation was to explore ways of combining our mutual interests of music composition, programming, computer music, and HCI in interesting ways that would lead to better interfaces and tools. The domain of computer programming was particularly interesting to us both as teachers and as practitioners. The use of sound in the debugging of programs is not new and dates back to 1950s and 1960s when programmers would tune AM radios to pick up the interference put out by the computer—they learned to monitor program behavior by listening to the radio [Vickers and Alty 2003]. James Alty had experimented informally with auralizations of the bubble-sort algorithm [Alty 1995] and it had become apparent that there was much scope for exploration.

The embryonic `CAITLIN` system was first presented at ICAD in 1996 [Vickers and Alty 1996] but there were some notable program auralization projects prior to this, viz., `InfoSound` [Sonnenwald et al. 1990], `LogoMedia` [DiGiano and Baecker 1992], `Sonnet` [Jameson 1994], `Auditory Domain Specification Language` [Bock 1994], and the `LISTEN` Specification Language (`LSL`) [Mathur et al. 1994]. Space does not permit a thorough comparison, but Table I highlights the main similarities and differences.

Author's address: Northumbria University, School of Computing, Engineering, and Information Sciences, Pandon Building, Camden Street, Newcastle upon Tyne, NE2 1XE, UK +44 (0)191 243 7614; email: `paul.vickers@unn.ac.uk`

Table I. Comparison of Program Auralization Systems

	InfoSound	LogoMedia	Sonnet	ADSL	LISTEN	CAITLIN
Domain of use	IC* Languages	Logo	Any language	C	Any language	Pascal
Type of auralizations employed	Music sequences and sound effects	Music pitches and sound effects	Music pitches and modulations	Music pitches and sound effects	Music pitches	Hierarchic musical motifs
Suitable for the novice programmer	No	No	No	No	No	Yes
Specialist knowledge required to define auralizations	Music composition	Some music and familiarity with MIDI	Some music and familiarity with MIDI	Some music and familiarity with MIDI	Some music and familiarity with MIDI	None
Extensible: auralizations can be programmed	Yes	Yes	Yes	Yes	Yes	No
Monomorphic: host language used in definition of auralizations	No	No	No	No	No	No
Uninvasive: source code left unchanged by auralizations	?	Yes	Yes	Yes	Yes	Yes
Employs preprocessing phase	No	N/A	No	Yes	Yes	Yes

2. RESEARCH PROCESS

When James Alty and I began the work there had been no formal or empirical study of program auralization. Researchers had focused on the technical novelty of mapping program data to sound [e.g., see Sonnenwald et al. 1990; DiGiano and Baecker 1992; Francioni et al. 1991; Jackson and Francioni 1994; Jameson 1994; Mathur et al. 1994] without formally evaluating its efficacy. In 1995 Bock [1995] reported a debugging experiment with his `Auditory Domain Specification Language`. However, the results were ambiguous (no control group) and equivocal (having only a 68% success rate). The earlier systems also relied on the user to define the required auditory mappings. A motivation behind our work was to see how auralization could support program comprehension and debugging for the general programmer population in which it is not reasonable to assume musical composition ability. The ICAD '96 paper presented results from a pilot study with a rough prototype of the `CAITLIN` system in which the auditory mappings were somewhat arbitrary. The structure of the auditory mappings was subsequently strengthened to follow a hierarchic organization with a formal tonal framework and was described in our ICAD '97 and ICAD '98 papers [Alty and Vickers 1997; Vickers and Alty 1998]. In those papers we made the case for using music as a communication medium.[1]

[1]Briefly, the reason is that the æsthetic forms of music (I do not just mean western tonal forms here either) have been developed over hundreds of years resulting in organizational schemes that are very effective for communicating ideas in multiple parallel streams. It seemed to us that the highly structured syntax of programming languages might map well onto a music syntax to give an effective auditory display.

Previous researchers demonstrated the technical feasibility of mappings from the programming domain to the auditory domain, but did not carry out any formal evaluation. Furthermore, the early systems tended to rely upon sound effects (auditory icons) and adhoc musical mappings—that is, the musicality of the auralizations depended on the ability of the programmer/user to specify mappings that adhered to conventions of music form and syntax. We recognised that whilst the ability to recognise a melody is innate[2] the ability to compose a melody (and especially the multiple melodies needed for a program auralization) requires training (and some innate skill too) and cannot be assumed. Therefore, an aim of the CAITLIN project was to design an auralization system with predefined mappings (motifs) that were organized around a common tonal framework, thereby enhancing the auditory ecology of the system. This, in turn, leads to greater potential for an effective and unambiguous auditory display in which the different musical streams complement rather than clash with each other. What the ICAD 2000 paper presented for the first time was empirical evidence in support of auditory displays for program comprehension and debugging tasks. Such evidence was lacking prior to this point, which is why we see the ICAD 2000 paper as an important milestone.

3. THE FUTURE

The ICAD proceedings series provides an historical account of the development and evaluation of the CAITLIN auralization system [also see Vickers and Alty 2002a; 2002b; 2003]. Having shown that musical program auralizations can be used in bug location and detection tasks, what remains to be done? The next developments in this field, as far as we see it, lie in three main areas: æsthetic issues of external auditory representations, their real-world usefulness, and how auditory representations can work with, support, complement, replace, and enhance visual representations.

3.1 Aesthetic Issues

The CAITLIN system reported in the ICAD 2000 paper used motifs structured according to principles of music grammar and cognition. As such, they had an æsthetic of western tonal music. The role of æsthetics in the design of computer artefacts and interfaces has recently begun to attract attention. At ICAD '04 in Sydney, Vickers [2004] discussed the role of æsthetics in the design of external auditory representations of programs (a more thorough treatment can be found in Vickers and Alty [2006]). The advantage of CAITLIN's fixed mappings is that they provide a coherent and self-consistent tonal framework. This ensures that the system's aural ecology is healthy and avoids the cacophony of other systems. The disadvantage is that a fixed æsthetic cannot be configured to suit different preferences and emotional or cultural needs. The next generation of auralization systems must take the issue of æsthetics seriously. Such a development could be considered to be extending the principles of literate programming [Knuth 1984]. Where literate programming tools of the past concentrated on typography and external visual representations to enhance presentation and comprehension of programs, the tools of the future can make use of auditory and musical æsthetics to extend the programmer's toolbox and visualization set.

3.2 Real-World Usefulness

To date, our experiments have been in the laboratory with undergraduate and postgraduate students. We need to go from the lab to the *real* programming world. To this end we have begun development of

[2]This observation is well supported by the music cognition literature.

Gamelan[3], a system for the auralization of object-oriented Java programs.[4] We will use Gamelan to explore the benefits of auralization in comprehending and debugging *real-world* object-oriented programs and their associated difficulties (e.g., locking in a multithreading environment).

3.3 External Auditory and Visual Representations

In addition to exploring the æsthetic, cultural, and real-world usefulness aspects of auralization it remains to see how auditory and visual mappings work together. We envisage that the temporal/spatial communication space provided by an audiovisual auralization system will provide a powerful set of tools to help programmers with writing, comprehending, and debugging their code. To take a simple example, we can imagine reading the source code or looking at a visualization of a data structure at the same time as listening to a number of threads passing messages and operating upon that data structure. This would be next to impossible with visual representations alone, as much switching of visual focus would be required [see Romero et al. 2002].

4. CONCLUDING REMARKS

The ease with which music and nonspeech audio can now be incorporated into programming environments (especially the Java platform) means that sophisticated spatial audio/visual tools are now even easier to build than when we began with CAITLIN project. Thus, we hope that more researchers will begin to explore this very interesting and rich avenue of enquiry.

REFERENCES

ALTY, J. L. 1995. Can we use music in computer-human communication? In *People and Computers X: Proceedings of HCI '95*, M. A. R. Kirby, A. J. Dix, and J. E. Finlay, Eds. Cambridge University Press, Cambridge. 409–423.

ALTY, J. L. AND VICKERS, P. 1997. The CAITLIN auralization system: Hierarchical leitmotif design as a clue to program comprehension. In *Proceedings of the International Conference on Auditory Display (ICAD '97)*. Xerox PARC, Palo Alto, CA. 89–96.

BOCK, D. S. 1994. ADSL: An auditory domain specification language for program auralization. In *Proceedings of the International Conference on Auditory Display (ICAD '94)*, G. Kramer and S. Smith, Eds. Santa Fe Institute, Santa Fe, NM. 251–256.

BOCK, D. S. 1995. Auditory software fault diagnosis using a sound domain specification language. Ph.D. thesis, Syracuse University.

DIGIANO, C. J. AND BAECKER, R. M. 1992. Program auralization: Sound enhancements to the programming environment. In *Graphics Interface '92*. 44–52.

FRANCIONI, J. M., ALBRIGHT, L., AND JACKSON, J. A. 1991. Debugging parallel programs using sound. *SIGPLAN Notices 26*, 12, 68–75.

JACKSON, J. A. AND FRANCIONI, J. M. 1994. Synchronization of visual and aural parallel program performance data. In *Auditory Display*, G. Kramer, Ed. Vol. XVIII. Santa Fe Institute, Studies in the Sciences of Complexity Proceedings, Addison-Wesley, Reading, MA. 291–306.

JAMESON, D. H. 1994. Sonnet: Audio-enhanced monitoring and debugging. In *Auditory Display*, G. Kramer, Ed. Vol. XVIII. Santa Fe Institute, Studies in the Sciences of Complexity Proceedings. Addison-Wesley, Reading, MA. 253–265.

KNUTH, D. E. 1984. Literate programming. *Comput. J. 27*, 2 (May), 97–111.

MATHUR, A. P., BOARDMAN, D. B., AND KHANDELWAL, V. 1994. LSL: A specification language for program auralization. In *Proceedings of the International Conference on Auditory Display (ICAD '94)*, G. Kramer and S. Smith, Eds. Santa Fe Institute, Santa Fe, NM. 257–264.

ROMERO, P., COX, R., DU BOULAY, B., AND LUTZ, R. 2002. Visual attention and representation switching during java program debugging: A study using the restricted focus viewer. *Lecture Notes in Computer Science 2317*, 221–235.

[3]Why, the native music of Java, of course!

[4]We note that Aditya Mathur has begun to develop his LSL system into this arena too, though he has yet to publish any results–see http://www.cs.purdue.edu/homes/apm/listen.html

SONNENWALD, D. H., GOPINATH, B., HABERMAN, G. O., KEESE, WILLIAM M, I., AND MYERS, J. S. 1990. Infosound: An audio aid to program comprehension. In *23rd Hawaii International Conference on System Sciences*. Vol. 11. IEEE Computer Society Press, 541–546.

VICKERS, P. 2004. External auditory representations of programs: Past, present, and future—an aesthetic perspective. In *Proceedings of the International Conference on Auditory Display (ICAD 2004)*, S. Barrass and P. Vickers, Eds. ICAD, Sydney.

VICKERS, P. AND ALTY, J. L. 1996. CAITLIN: A musical program auralization tool to assist novice programmers with debugging. In *Proceedings of the International Conference on Auditory Display (ICAD '96)*, S. P. Frysinger and G. Kramer, Eds. Xerox PARC, Palo Alto, CA. 17–24.

VICKERS, P. AND ALTY, J. L. 1998. Towards some organising principles for musical program auralization. In *Proceedings of the International Conference on Auditory Display (ICAD '98)*, S. A. Brewster and A. D. N. Edwards, Eds. Electronic Workshops in Computing. British Computer Society, Glasgow.

VICKERS, P. AND ALTY, J. L. 2002a. Using music to communicate computing information. *Interacting with Computers 14,* 5, 435–456.

VICKERS, P. AND ALTY, J. L. 2002b. When bugs sing. *Interacting with Computers 14*, 6, 793–819.

VICKERS, P. AND ALTY, J. L. 2003. Siren songs and swan songs: Debugging with music. *Commun. ACM 46*, 7, 86–92.

VICKERS, P. AND ALTY, J. L. 2006. The well-tempered compiler: The aesthetics of program auralization. In *Aesthetic Computing*, P. Fishwick, Ed. MIT Press, Boston, MA. Chapter 11, in press.

Received February 2005; revised June 2005; accepted July 2005