LSL: A Specification Language for Program Auralization

Aditya P. Mathur, David B. Boardman, and Vivek Khandelwal
Software Engineering Research Center and Department of Computer Science
Purdue University
W. Lafayette, IN 47907
Email: apm@cs.purdue.edu (317) 463-7822

Abstract

The need for specification of sound patterns to be played during program execution arises in contexts where program auralization is useful. We present a language named LSL (**L**isten **S**pecification **L**anguage) designed for specifying program auralization. Specifications written in LSL and included in the program to be auralized are preprocessed by an LSL preprocessor. The preprocessed program when compiled and executed generates MIDI or voice data sent through a MIDI interface to a synthesizer module, or via audio channels, to an audio processor, which is transformed into audible sound. LSL has the generality to specify auralization of a variety of occurrences during program execution. It derives its broad applicability from its few generic elements, that when adapted to any procedural programming language, enable the use of LSL specifications for auralizing sequential, parallel, or object oriented programs. We view LSL as a useful tool for building general purpose multimedia applications, auditory displays, and for research in program auralization.

1 Introduction

The idea of using sound to understand program behavior and to analyze data using sound has been reported by several researchers. By program and data auralization we refer to, respectively, the activity of mapping aspects of program execution or properties of data to sound. Human voice and digitized sounds are included in this context. A survey of the use of sound in understanding program behavior or analyzing data collected from experiments or generated by a program is reported by Francioni and Jackson [5]. It is believed, and has been demonstrated in a few cases, that the use of sound can enhance program understanding. Yeung [10] proposed the use of sound to recognize multivariate analytical data. Francioni and Jackson [5] used program auralization to understand the runtime behavior of parallel programs. Brown and Hershberger [3] auralized some of their animations generated using the Zeus animation system. In his doctoral dissertation, Edwards [4] built and evaluated a word processor with an audio interface for use by visually handicapped users. Gaver [6, 7] proposed the use of auditory icons for use as a part of Apple's interface on the Macintosh machines.

We have designed a language that simplifies the task of specifying which occurrences during program execution are to be auralized and how. The language is named Listen Specification Language, abbreviated as LSL. Listen is the name of our project to investigate possible uses of sound in programming environments and software based applications.

The need for LSL, its syntax, use, and current implementation status are reviewed in the remaining sections. Details of LSL syntax and an LALR(1) grammar, appears in [2]. Section 2 outlines the need for specifications of program auralizations and LSL. Language requirements

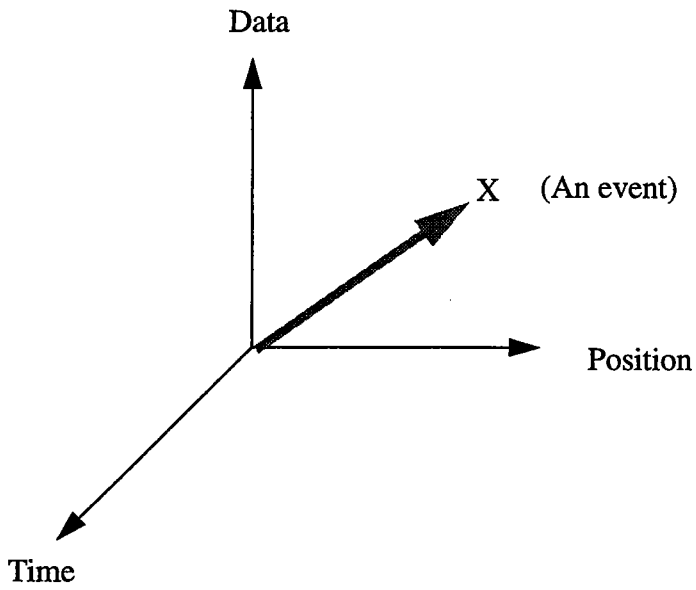

Figure 2: Occurrence space characterization in LSL.

Position refers to any identifiable point in a program. For example, in a C program, beginning of a function call, end of a function return, beginning of a while-loop, beginning of a while-loop body, and beginning of a condition, are all positions. In general, an identifiable point is any point in the program at which an executable syntactic entity begins or ends. This implies that a position cannot be in the middle of an identifier or a constant. In terms of a parse tree for a given program, any node of the parse tree denotes a position. For example, the subscripted dot ($\bullet_i$) denotes seven possible positions in the following assignment: $\bullet_1 X \bullet_2 = \bullet_3 X \bullet_4 + \bullet_5 3 \bullet_6 / \bullet_7 2$.

Data in a program refers to constants allowed in the language of the program being auralized and the values of program variables. A *data relationship* is an expression consisting of constants, variables, and function calls. Time refers to the execution time of the program. It is measured in units dependent on the system responsible for the execution of the auralized program. In an heterogeneous system, time is measured in units agreed upon by all elements of the system.

As shown in Figure 2, a three dimensional space is used for specifying occurrences in LSL. Two kinds of occurrences are distinguished: events and activities. LSL allows an arbitrary combination of data relationships, positions, and time to specify an event or an activity associated with program execution.

3.3 Sound Space Characterization

The sound space is characterized by sound patterns comprised of notes, durations, play styles, and instruments. Notes of arbitrary durations can be combined to form sound patterns. Each note can be associated with one of several play styles and with an arbitrary instrument. For example, a note can be played staccato on a piano with a specified volume. Combining notes in various ways gives rise to a domain consisting of an infinity of sound patterns. Digitized sound, such as human voice, is considered a sound pattern.

```
begin auralspec
specmodule call_auralize
var
     gear_change_pattern, oil_check_pattern, battery_weak_pattern: pattern;
begin call_auralize
     gear_change_pattern:= "F2G2F2G2F2G2C1:qq"+ "C1:f";
     oil_check_pattern:= "F6G6:h";
     battery_weak_pattern:= "A2C2A2C2";
     notify all rule = function_call: "gear_change" using gear_change_pattern;
     notify all rule = function_call: "oil_check" using oil_check_pattern;
     notify all rule = function_call: "battery_weak" using battery_weak_pattern;
end call_auralize;
end auralspec.
```

Figure 3: Auralizing an automobile controller.

3.4 Language Independence

The second requirement stated above is significant as we want LSL to be usable by programmers regardless of their preference for a given other programming language. Adherence to this requirement has produced a language which in the strict sense should be considered as a meta-language. One can therefore adapt LSL to specific programming languages as described later. In examples below, we use a C adaptation of LSL termed LSL/C.

4 Examples of LSL/C Use

We now present two examples illustrating some aspects of LSL. Each example consists of a problem statement and a solution using LSL/C. In each example we assume that, unless specified otherwise, default values are used for various sound related parameters such as midi channel, timbre, volume, and metronome value.

Example 1

An automobile contains a distributed microcontroller network. The software to control this network and other automobile functions is to be tested and debugged. All calls to functions gear_change, oil_check, and weak_battery are to be auralized by playing suitable sound patterns. Figure 3 exhibits an LSL/C specification module that meets this auralization requirement. In this example, each `notify` command specifies an event using a `rule` named *function_call*. The sound to be used for event notification is specified with a `using` clause within a `notify` command. The `all` prefix indicates that all occurrences of the specified function call are to be auralized. The `selective` alternative, not elaborated here, specifies auralization of selective occurrences of a rule.

Example 2

This example illustrates how `track` can be used with VDAP (Value Dependent Aural Pattern) to track arbitrary functions of program variables. Suppose that it is desired to track the dynamic

[4] Edwards, A. D. N. "Soundtrack: An Auditory Interface for Blind Users." *Human-Comp.r Interaction* **4(1)** (1989): 45–66.

[5] Francioni, J. M., and J. A. Jackson. "Breaking the Silence: Auralization of Parallel Program Behavior." Technical Report TR 92-5-1, Computer Science Department, University of Southwestern Louisiana, 1992.

[6] Gaver, W. W. "Using sound in Computer Interfaces." *Human-Comp. Interaction* **2** (1986): 167–177.

[7] Gaver, W. W. "The Sonicfinder: An Interface that Uses Auditory Icons." *Human-Comp. Interaction* **4(1)** (1989): 67–94.

[8] Langston, P. S. "Little Languages for Music." *Computing Systems* **3(2)** (1990): 193–282.

[9] Thompson, T. "Keynote—A Language and Extensible Graphic Editor for Music. *Computing Systems* **3(2)** (1990): 331–358.

[10] Yeung, E. S. "Pattern Recognition by Audio Representation of Multivariate Analytical Data." *Analytical Chemistry* **52(7)** (1980): 1120–1123.