BITS C461 Software Engineering

First Semester 2003-2004

Project Requirements:

JListen: A Program Auralizer for Java

Project Sponsor: Aditya P. Mathur

Department of Computer Science

Purdue University, West Lafayette, IN, USA

Original: Aug 4, 2003. Revision: None.

1 Long Term Goal

The long term goal of this project is to create a commercially available, open source, too for auralizing Java programs. When executed, an auralized Java program is able to send commands to a sound server which then obeys these commands thereby resulting in sounds. Auralization of Java programs is likely to be useful in debugging, announcement of special events such as intrusion attempts, and for pure entertainment.

For the purpose of auralization, the execution of every program is considered as a sequence of events and activities. For example, "the value of a program variable become 0" is an event. The execution of a loop inside a program is an activity that covers several events. It is these program events that are mapped to various sounds through the auralization process. One can hear tthe auralized program during its execution and conclude what events have occurred and what are the ongoing activities.

2 What is to be done?

Selected teams of students of BITS C461 will be required to complete the following tasks related to this project:

1. Understand and analyze project requirements. The product of this activity will be a list of high level and low level use cases, system sequence diagrams, and a domain model expressed in UML.

2. Complete a design of the various components of the Jlisten. The product of this activity will be a complete design of JListen and its individual components. The design must be expressed in UML.

3. A prototype of Jlisten. The prototype will demonstrate how a full-fledged JListen would work. The prototype will have only a selected subset of the features that a full fledged JListen might have. Teams working on the project will decide what features to select for prototyping.

3 Using JListen

As shown in Figure 1, a user of Jlisten begins with a Java program *J* that is to be auralized. The user then specifes what events and activities in *J* are to be auralized. This specification is expressed in Listen Specification Language (LSL). LSL allows specification of events and activities in a program that need to to be auralized. LSL also allows the specification of what sounds are to be associated with an event or an activity and at what rate these are to be played. The core of LSL is independent of the programming language used for auralizing a program. By adding a list of keywords to the LSL core, one obtains LSL/L for programming language L. For example, LSL/Java is obtained by adding Java keywords to LSL core. Additional information of LSL is available from the following site.

http://www.cs.purdue.edu/homes/apm/research–¿Program Auralization

An LSL specification and *J* are input to an LSL parser and then to Java Program Auralizer (JPA). The auralizer generates an instrumented version of *J*. The instrumented version of *J* contains calls to a Listen Sound Server (LSS). The instrumented *J* is then compiled using any Java compiler to produce an executable version of *J*. During execution, whenever a call to LSS is executed, LSS receives a request to play a sound. LSS uses its current parameter settings to decide whether or not to play the sound, and, if the sound is to be played then on which audio device should it be played.

LSS is a stand alone component of JListen. It is independent of any programming language. Thus, for example, one could use LSS for processing requests coming from a C or a C++ program.

4 LSS: Listen Sound Server

This extremely powerful stand alone application communicates with any auralized application and processes incoming requests for generating sounds. An auralized application may communicate with LSS in a variety of ways. It could also be compiled with the auralized application. The LSS can be executing on a machine different from

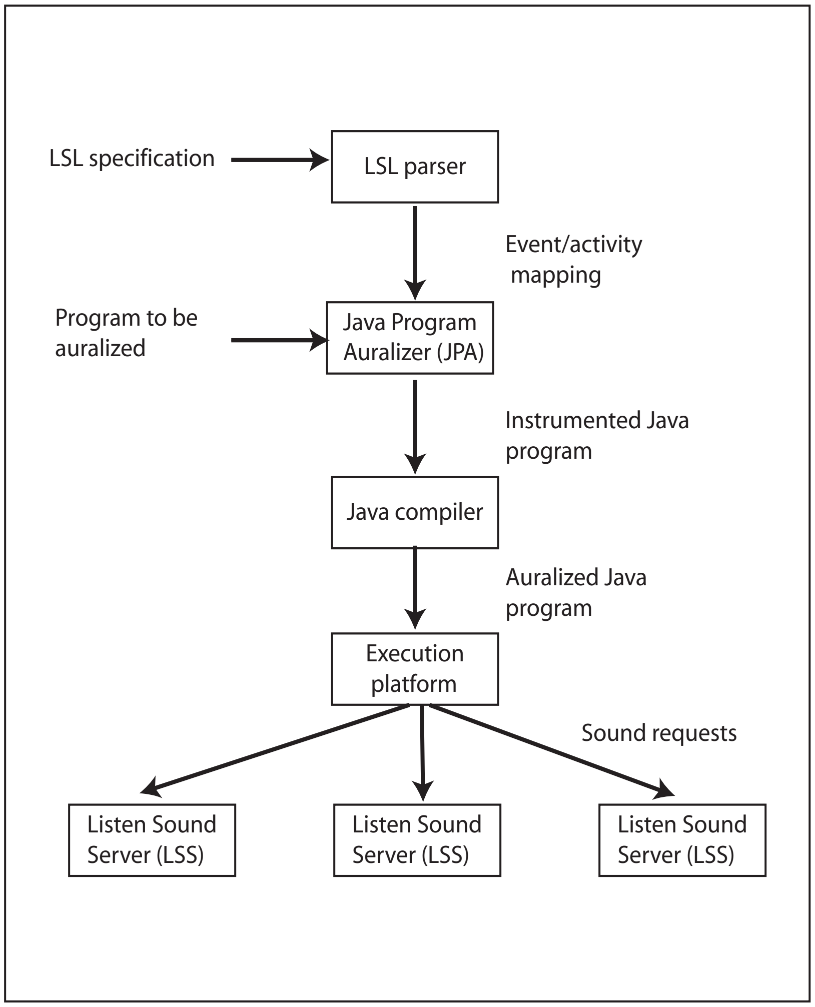

Figure 1: Interactions amongst components of JListen. Three instances of LSS are shown. In practice the number of LSS instances could be zero or more.

the machine on which the auralized program executes. This feature allows a single auralized program to be heard at any computer accessible via the network.

LSS allows a user to selectively listen for zero or more events and activities that have been auralized. Thus, for example, an LSL specification might call for events E_1 and E_2 in program J to be auralized. However, any time during the execution of J, a user could instruct LSS not to play the sound corresponding to evgent E_1. Further, the mapping between an event and a specific sound can also be altered during the execution of J. For example, if the LSL specification calls for playing a note on the Sitar whenever event E is detected, a user could change this mapping of E to Sitar to E to Banjo.

LSS can also receive audio requests from more than one auralized application. Thus, for example, auralized programs J_1 and J_2 could send audio requests to LSS concurrently. The user can tune LSS to play any of the programs, none of them, or both of them simultaneously. Thus, LSS serves as a multichannel concurrent radio for auralized programs.

The requirements given here are incomplete and ambiguous. Teams will need to resolve ambiguities and, where necessary, complete the requirements. Help wil be available from the instructors.