Rivendell
Design Overview
26 April 2002

The overall structure of the Rivendell system is envisioned as being an interconnected system of software components. Depending upon the size of the facility served and degree of redundancy required, these functional blocks will be able to work in a wide variety of hardware configurations, from a single computer to a large, LAN connected cluster. Communications between these components will be by means of TCP/IP connections.

Within the context of the application, the software components can be divided into two broad categories: the *core* layer and the *presentation* layer. As the names imply, the *core* layer is where the primary functionality of the application resides, while the *presentation* layer is primarily responsible for presentation of an interface to the user.

Core Layer

This layer consists of the following components:

Audio Storage
A storage medium for the digitally encoded audio, this could range from a single hard drive to a fibre−channel disk array, depending upon the needs of the facility. A noteworthy feature of the design is that this storage is *not* generally available to all the components of the application, but is only accessible from the Core Audio Engine. This architecture gives Rivendell a strong mechanism for centrally managing bandwidth utilization within the audio storage subsystem.

RDBMS
A relational database management system, used to record and track the details and status of all audio and events, thus becoming the central bookkeeper of the system. Rivendell will use the well known **MySQL** Open Source RDBMS.

Core Audio Engine
The very heart of the Rivendell system, this component operates the actual audio and communications hardware (such as audio adapters, serial and GPI interfaces) and is responsible for all audio playout and recording. No other components in the system directly touch these resources. The hardware abstraction provided by this component is at a fairly low level, being somewhat analogous to that of a tape recorder with the functions **PLAY**, **STOP**, **REC**, etc.

Playout Engine
This is where audio and data meet. The Core Audio Engine has no knowledge or awareness of the database −− it only plays or records specific cuts when told to. The Playout Engine implements the abstractions of a modern digital audio system −− features like playlists, netcatcher schedules, production recorders and the like −− on top of the basic "tape recorder" interface provided by the Core Audio Engine. It also provides a TCP−based command and control channel to allow user clients to connect and control its operations.

The above components themselves constitute a functional digital audio system, in the sense that a log could be loaded into it, placed into **AUTO** mode and let play. In theory, no user interface at all would be required for this. Of course, in any practical system, user interfaces *are* required, and Rivendell makes full provision for them in the *presentation* layer. The reason for seperation of the two functions is worth noting here, however. Historically, in any application, it is the user interface portion that is most prone to bugs and instability. The separation of the core application components from the user interface thus provides a significant measure of insulation for the former from user interface bugs. Thus, bugs or a crash in the presentation layer systems need not interrupt proper execution of the application within the core layer.

Presentation Layer

The choice of MySQL and TCP control for the Playout and Core Audio Engine components leaves us with lots of choices in the *presentation* layer. In theory, it would not be difficult to implement control clients using a wide array of platforms, such as Windows, Linux or Mac OS X. In practice, two basic methodologies for implementing control clients are envisioned:

X11/VNC Client

These clients could be run either on individual computers located at the various work locations (control room, production room, etc) or run on one or more central *GUI Servers* whose output would be directed through the network to *thin clients* located at the work locations. The use of thin clients would be particularly compelling for larger facilities, in that these terminals are rugged, inexpensive, have no moving parts such as fans or disk drives (and hence are quiet) and need no local configuration whatsoever. Planned X11/VNC clients include the following:

Production Room Interface
: For recording and audition of spots and other audio on the system

Netcatcher Interface
: For configuring the automatic recording features of the system.

Simple Control Room Interface
: A "generic" on−air interface.

We will certainly be adding others to this list as discussion goes forward.

Windows

A traditional program written for Windows is quite viable as a control interface too, particularly where there are legacy systems (such as traffic packages) to interface with.

Traffic Interface
: For interfacing with traffic and music scheduler systems.

The above is only a broad outline. As development goes forward, this model will doubtless be honed and refined. Rivendell aims to be above all a *user driven* system, and so will only be as good as the ideas and input from folks in the "real world".