An HTTP Interface to Common Music

Heinrich Taube
School of Music, University of Illinois
taube@uiuc.edu

Tobias Kunze
CCRMA, Stanford University
tkunze@stanford.edu
http://www.stanford.edu/~tkunze

Abstract
This paper describes the addition of web-compatible network support to Common Music for a large subset of its functionality. The core score description language and most output capabilities reside on a serving machine and communicate via a native LISP HTTP server with any authorized machine in the local or global network. Accessing the system via the network protocol is intended to enhance existing functionality rather than replace local interaction with the software. The choice of HTTP and its related protocols is implied by the goal of providing a composition server. The wide availability of web browsers and other clients as well as their inherent platform-independency and support for executable code (Java) set web-based access methods well ahead of other options. Adding server functionality to the composition software relieves weaker machines from running a full-fledged LISP environment and enables all clients to make use of scarce or non-standard resource on the server side, such as platform-specific target software or sound synthesis capabilities. In addition, a client-server design of composition software facilitates greatly lab administration and classroom teaching. Although moving the system in the direction of a client-server architecture seems desirable from a user’s or administrator’s point of view, we expect composers to benefit most from the ease with which other software clients will be able to use Common Music’s compositional services.

1 Composition Server Model

1.1 Motivation
Common Music is a portable composition environment that is able to control a number of different synthesis languages. The system is implemented in Common LISP and C and runs on most computers available today. Though Common Music is highly portable, its kernel must be customized to particular machine configurations when the software is installed. Until recently the portable nature of the software necessitated a number of compromises in system functionality and support for any given local configuration.

The main purpose of this project is to leverage recent HTTP developments to solve local configuration constraints inherent in the portable nature of Common Music. In some sense, this move to a client-server architecture is a natural outcome of Common Music’s central aims of portability and synthesis independence. By shifting some of CM’s features to HTTP, a number of important features are gained. In particular, it

- relieves weak machines from constraints imposed by local configurations
- gives access to expensive resources such as synthesis services
- allows audio, MIDI, graphics, etc. to be streamed across the network in realtime
- provides a uniform GUI interface
- facilitates classroom teaching
- facilitates system administration.

1.2 Stand-Alone Configuration Constraints
Limits of the local host affect Common Music in several areas. With respect to implementation, the software’s portability depends on a number of different, unequally featured Common LISP implementations. For example, some implementations provide CLOS, some do not; some provide an interface to a native windowing system, others do not; some have native compilers while others byte compile; some provide dynamic foreign function loading and others do not. The net effect of the uneven implementation features is that no single machine supports a “fully-featured”
Table 1: Example CM feature support across five local configurations.

<table>
<thead>
<tr>
<th></th>
<th>GUI</th>
<th>MIDI Driver</th>
<th>MIDI File</th>
<th>CLM</th>
<th>Csound</th>
<th>MusicKit</th>
<th>RT</th>
<th>HTML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mac OS</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>Windows ’95</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SGI IRIX</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>NeXTStep</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>Linux</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td></td>
</tr>
</tbody>
</table>

1.3 Accessing Synthesis Services

The host computer also affects the set of synthesis options a locally configured Common Music image will control. Although the system supports many different synthesis languages, a composer is actually limited to those languages that can run on the local CPU. Luckily, there are several good synthesis packages (such as CLM and Csound) that are also highly portable. But the fact remains that there are many more computers that can run compositional algorithms with adequate speed than there are computers suitable to executing synthesis algorithms. Moving to a client/server model means that a composer can work with synthesis languages and hardware resources that are not available on the local host. In this model, a remote server equipped with a fast CPU and adequate memory renders sound and returns audio or MIDI streams, or general control information to the local client’s plugins to play. The availability of audio and MIDI streaming in real-time also means that CM no longer needs to provide some of its scheduling and MIDI driver support currently required for each local port.

1.4 Towards a Uniform GUI

The most severe penalty for making CM a portable system has been its relatively weak support for GUI tools. The primary cause of this is the ANSI Common LISP Standard itself, which does not address GUI issues at all. This means that to provide a “portable” LISP-based graphical interface, a developer must completely reimplement the interface for every machine/LISP combination. For this reason, Common Music’s GUI (Capella) currently runs only on the Macintosh. But the past few years have witnessed a explosive growth in HTTP GUI development; at this point, HTTP support is more active and broad-based than that provided by Common LISP vendors. While HTTP graphics is still rather primitive, its central emphasis on hypertext-based presentation is consistent with the description of algorithmic processes in Common Music. In fact, system documentation, tutorials, dictionaries and ancillary documentation have already been in HTML format for several years. It is now a natural step to extend this support into the presentation of the system itself.

1.5 Inter-Application Services

Providing a simple, network-oriented public interface to Common Music also enables other software clients to make use of its services and thus to enhance their functionality.

Although by no means a full-fledged Inter-Application Communication (IAC) protocol, HTML combines basic IAC abilities with ease-of-use, network transparency, and a large existing base of media-oriented software components, including web access libraries—features that are desired in today’s highly heterogeneous software worlds. For example, Cecilia currently offers Cybul for providing algorithmic score production, a language similar to, albeit less expressive than Common Music. A production system such as this could benefit greatly from an evaluation mechanism for Common Music services via HTTP as for its tk-based graphical editor.

1.6 Teaching Experience

Another goal of the composition server project is to support courses in algorithmic composition. Over the past few years courses at CCRMA and UIUC have resulted in a collection of HTML presentations of algorithmic topics such as randomness, pattern description, chaos, iterative functions and so on. The server

\[1\text{Cecilia is a music/sound production system that uses MIT’s Csound as its sound-processing language (see http://www.musique.umontreal.ca/Org/CompElectro/CEC/).}\]
model allows the functionality of these documents to be extended into interactive, structured sessions with the server, without the necessity for a student to first master the implementation issues of the underlying system. For example, the HTML pages explaining weighted random selection allow the student to experiment with the effects of changing weights in compositional material using a simple table display. When the student selects GO, the contents of the table is sent to a corresponding algorithm object in the kernel which renders the choices and returns the results back to the student in the form of a MIDI stream.

2 Implementation

A Common Music system configured to act as a composition server differs from a stand-alone image in several respects. First, it relies on the services of CL-HTTP, a LISP-based HTTP server, developed and maintained at MIT. Secondly, it contains a new HTTP subsystem that controls the HTTP server and manages distinct user sessions, authenticating HTTP requests and restricting access to features of the Common Music server image as desired. For security reasons, the server image should reside in a private, secured area of the server’s filesystem (cf. Figure 1).

2.1 The LISP HTTP Server

Using an internal, LISP-based HTTP server as opposed to a stand-alone, external server has proved to be advantageous for a variety of reasons. Foremost, data that is to be passed back and forth between the server and Common Music need not to be copied, since both share the same application context; also, no data conversion has to take place, since both are LISP software modules. As a result, data exchange between a native, resident server and Common Music is highly efficient, both in terms of speed and memory usage. As a LISP program, the server may also be easier configured and tuned in run-time and under LISP program control by altering its private data structures. This, for instance, makes it easy to add temporary custom MIME type translations or regulate user access to the server. From an administrative point of view, dedicating a separate HTTP server to Common Music permits only these requests to be routed via a different network port, thus contributing significantly to the overall security of the server.

2.2 Session Management

The most difficult problem in making Common Music’s services available to a number of clients is the single-user design inherent in the LISP interpreter. However, although concurrent execution is still not part of the upcoming ANSI Common LISP standard, most implementations today support some notion of it and do provide basic facilities to construct a multi-user environment, such as multiple processes and system globals for user home directories. Common Music’s HTTP subsystem uses these facilities, when available, to simulate a ‘cooperative’ multi-user environment that shields user contexts, called sessions, as much as possible from each other. In particular, by using standard LISP packages and multiple processes, it maintains a different name space, file system area, and process group for each active session.

A session is established, when a user logs in and automatically times out after it has been idle for a configurable time. Once the user has gained authentication, subsequent HTTP requests are automatically assigned to its session, subject to site-specific restrictions. (A Common Music server image running on a remote server, for example, may restrict the use of its serial ports for MIDI to local users only.)

However, evaluation requests submitted via HTTP may contain programming errors or other constructs that bring the LISP system to a halt. As a result, the HTTP subsystem monitors sessions continuously in order to detect, and eventually abort, runaway processes or infinite loops.

Finally, the multi-user environment is also cooperative in the sense that system resources such as memory or disk space are shared among all user sessions, with no concept of quota or priority imposed.

2.3 Security

Lisp is essentially an open-ended development environment. As such, it provides unauthenticated access to the host machine limited only by whatever restrictions the operating system imposes on the user. In fact, part of the power of LISP as a development environment stems from the generality with which it interfaces to traditional operating system services—a generality and ease-of-access that may seriously compromise the host machine’s security. On machines where security is of concern, the Common Music Server should reside in a private, separate area of the file system, similar to an anonymous Internet ftp server.

When turned into a multi-user environment, LISP is also insecure internally: no read- or write-permissions protect the system from user processes...
or user processes from affecting each other. Clearly, providing separate name spaces and process groups does not fully protect a session from being affected by others. Specifically, system globals, system packages, or Common Music’s toolbox, but also CLOS definitions cannot be made read-only without crippling the system and thus seriously jeopardizing its overall usefulness. However, since full session protection within a single LISP system may not be achieved without giving each session a copy of the complete LISP environment—thus essentially reduplicating the LISP image for each user—they have been left unprotected in the current design. As a result, user processes may affect each other not only actively by invading another user’s package, but also inadvertently by side-effecting data in the system area.

2.4 Future Developments

Although the server’s internal vulnerability does not appear to have much impact on its usability in anticipated typical (classroom) situations, it is recognized as a serious design flaw and as a problem affecting the maintenance of the server. A short-term goal is thus to devise a mechanism that adds full write-protection to all system and foreign user packages.

Long-term goals include the addition of a LISP-to-JavaScript translator to the HTTP subsystem to enable remote execution and synchronization of Common Music output such as MIDI and graphics display.