Wirelessly Interfacing with the Yamaha Disklavier Mark IV

Matthew Teeter and Christopher Dobrian
University of California, Irvine, United States, dobrian@uci.edu

Abstract — **The music technology industry is only recently beginning to realize the potential of wireless communication technology for control and communication of data for music and multimedia applications. A new breed of musical devices is starting to integrate technology that allows the wireless transmission of MIDI (Musical Instrument Digital Interface) messages, real-time audio and video data, control data for performance synchronization, and commands for remote hardware control of these instruments. The Yamaha Disklavier Mark IV piano, which debuted in 2004, is the first instrument with wireless capabilities built-in [1]. It communicates via the 802.11b protocol (WiFi), which allows the piano to transmit and receive information to/from nearby wireless controllers. The piano originally comes with two such controllers: the handheld Pocket Remote Controller (PRC), as well as the larger Tablet Remote Controller (TRC). Both of these devices are proprietary, closed systems that accomplish the specific function of controlling the piano. In this project, we wished to create platform-independent software having the same functionality as these existing controllers, which could run on a conventional laptop with wireless capabilities. Although this solution has several advantages over the prepackaged solutions, it is unsupported by Yamaha because it was developed entirely at the University of California, Irvine. We were able to interface with the Disklavier by sniffing wireless network traffic with Ethereal [2]. We then deciphered this raw information to determine the messaging protocol the Disklavier used to communicate with the supplied controllers. Once we understood the inner workings of the piano, we created software using a variety of technologies, including Java, PostgreSQL, XML, and Flash. Our software can control a Mark IV piano from a Windows, Mac, or Linux laptop, thereby opening new possibilities in music creation and performance. Although we assume the primary users of our software will be universities, we hope it benefits the music technology industry as a whole.**

BACKGROUND

Controlling instruments wirelessly opens many new possibilities regarding aesthetics, concert atmosphere, and audience interactivity. In typical computer music concerts, the instrument gets its instructions from one computer that is connected with cables to the MIDI In/Out ports. In a wireless situation, however, any number of computers could communicate with the instrument without the need for re-cabling, allowing for far greater levels of convenience and interactivity.

Wireless control of instruments has other benefits as well. A common occurrence at computer music concerts is that listeners are often distracted by the myriad cables surrounding the performers on stage. Instead of concentrating on the music, they might be tempted to think about the technology involved in controlling the instruments, or watch the blinking lights on the mixer, etc. Wireless communication in a musical setting helps to remedy this problem. Eliminating cables and thereby hiding some of the technological aspects serves to make the computer-controlled piano seem less like a newfangled contraption and more like a conventional instrument people are familiar with. These are just two ideas of ways in which wireless technology can improve the concert atmosphere.

We hope others will continue to explore this area further using our software. To conform to Yamaha's controller naming scheme, we decided to dub our software the Disklavier Laptop Remote Controller (DLRC). We shall hereafter refer to it using this name in order to avoid confusion with the original controllers. Despite the name, a laptop is not required - the software could run on any device that supports Java, ability to access a PostgreSQL database such as JDBC (Java Database Connectivity), and Flash (which is actually only required for the user interface). Note that because controlling the Disklavier from a PC deviates from Yamaha's Terms of Use, Yamaha is not liable for any problems encountered while using our software.

PREVIOUS WORK

Ever since the Mark IV Disklavier debuted in 2004 [3], there was no way of controlling the piano from a third-party device before we created DLRC. We found one report of someone using a PDA-like device to control MIDI equipment wirelessly [4]. In Phil Dayson's system, a PDA with a wireless card was used to send commands to a computer that was wirelessly connected on the same network. The computer then sent MIDI information to a keyboard. A system similar to this one is already in place with the Disklavier Mark IV, except that the Disklavier's PDA device, the Pocket Remote Controller, communicates directly with the piano itself. The Disklavier system thus removes an intermediate step in the line of communication, thereby reducing latency. Nevertheless, having to use the supplied controllers was constraining, and we wanted to eliminate this problem by controlling the piano with a standard (non-proprietary), readily-available laptop.

Other companies, such as M-Audio and CME, are beginning to realize the potential that wireless capabilities can have on the digital music world. Both companies offer wireless MIDI adapters so that MIDI cables can be eliminated. This is perfect for musicians who like to move around on stage while playing. Although the Disklavier does not yet come with a wireless MIDI system, we

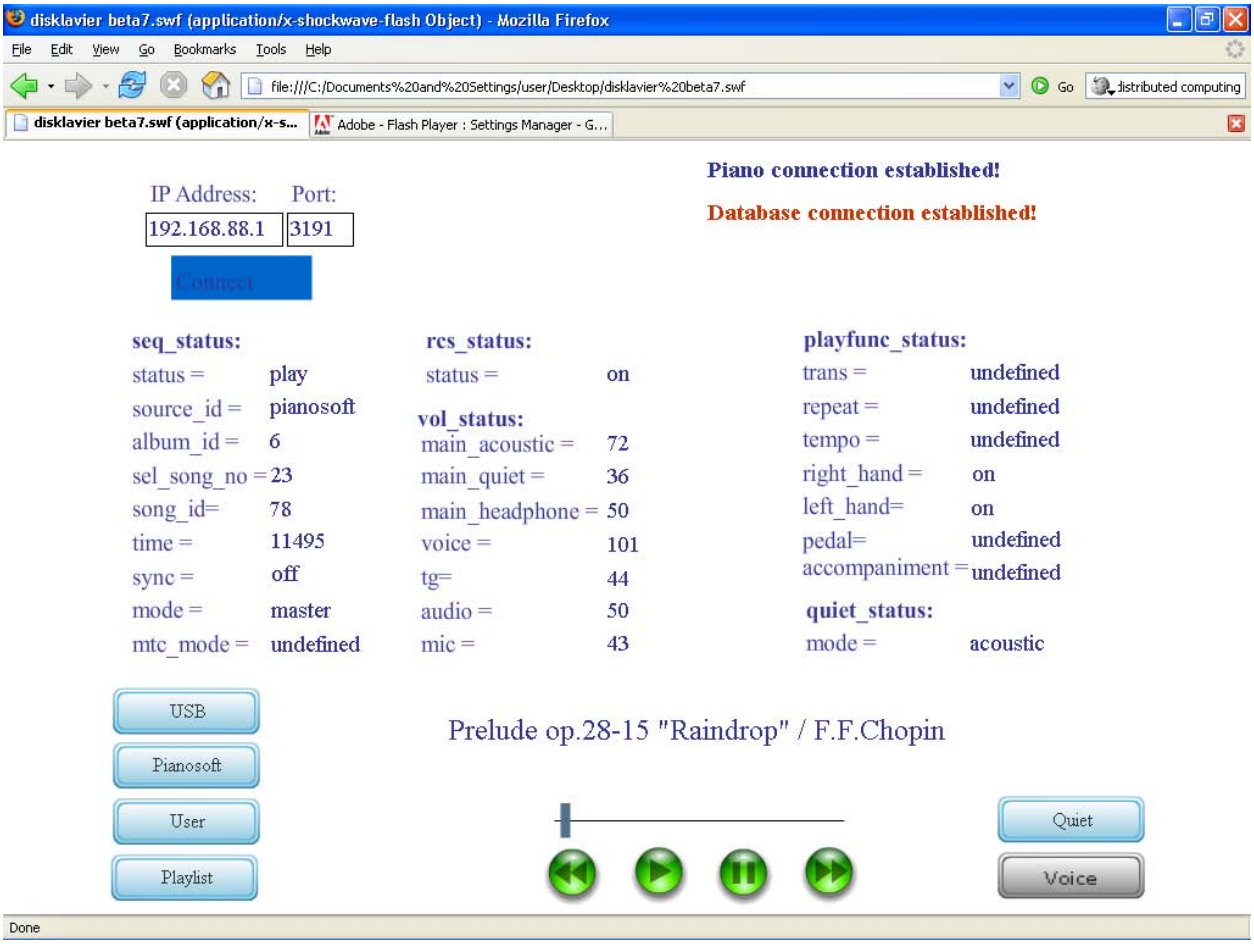

Figure 1: Our software working in a standard web browser.

anticipate such systems will become commonplace in the future.

MOTIVATION

The DLRC provides many benefits over the existing controllers. First of all, it mitigates the risk of losing or damaging an existing controller. Because both the PRC and the TRC are quite costly to replace, non-proprietary, free software is certainly valuable. Besides the risk-mitigation benefit, having an easy-to-replicate controller is quite useful in other ways. For example, consider a university setting where many students and teachers use the Disklavier at different times. To keep track of the controllers, users must check out the controller equipment before each use, and return it when finished. This is obviously somewhat inconvenient; the problem can be eliminated if users could control the piano from their own laptop, thus avoiding the check-out process altogether.

The DLRC also contains software improvements over the existing controllers. For example, our software allows users to more easily navigate to and select songs from their USB drive. The original PRC software displays all folders on the USB drive, regardless of whether they have playable MIDI files within them or not. In the v1.2 firmware, all subdirectories on the drive are displayed as their own album. If the user selects an album that contains no songs, a "No songs" message is displayed and the user must navigate back to the album menu to try again. Our software avoids this inconvenience by only displaying albums that have playable files within them. We were able to do this by simply modifying a SQL query used to access the songs on the USB drive. Considering that the average person has many types of files stored on their USB drive besides only MIDI files, this feature vastly improves the USB navigation interface.

We also added capabilities which the original software could not perform, such as slowing tempo down to less than 50%. This could be useful when attempting to learn a song with a fast tempo. Furthermore, users can more rapidly select songs with our interface, because more are displayed on the selection screen at once (see Fig. 2). These functionalities, along with others, such as the ability to automatically bring the Disklavier out of standby when the DLRC is started, greatly simplify the user experience.

Finally, because the DLRC was written in a manner that maintains compatibility with the existing controllers, any number of laptops, PRCs, and TRCs can be connected simultaneously to the same piano. This enables new possibilities in music performance and audience interaction. For example, consider an interactive concert involving a Yamaha Disklavier piano connected wirelessly with laptops in a concert setting. An audience could bring in multiple laptops and simultaneously send input to the piano, influencing the music in some way while sitting in their seats and quietly listening!

A thorough description of our software, along with a description of the piano's software configuration, follows.

SYSTEM OVERVIEW

The Disklavier consists of a standard acoustic piano outfitted with mechanisms to act as a player piano, with built-in speakers, and a passively-cooled Linux computer

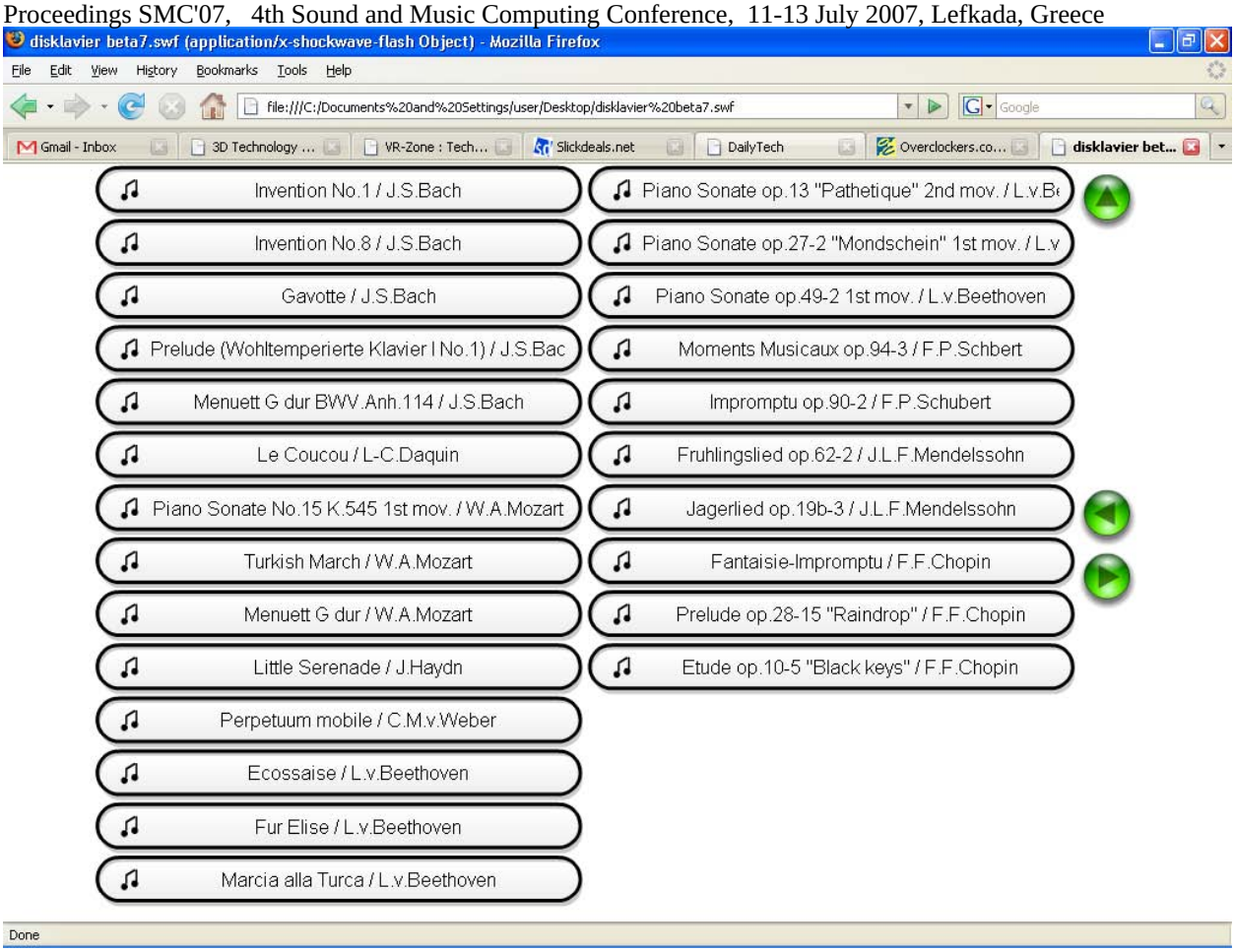

Figure 2: The DLRC song selector displays 28 songs per screen, as opposed to the TRC's ten or the PRC's six.

equipped with a wireless router. A Disklavier thus allows nearby wireless devices to connect to its built-in computer and communicate with the piano by using IP addresses assigned using DHCP. The pianos come with a factory-default IP address of 192.168.88.1. Controllers connected to the piano are assigned IP addresses on the same subnet as the piano (e.g. 192.168.88.7). Each controller must open one TCP connection on port 3191 to exchange information with the Disklavier server software and another connection on port 5432 (the default PostgreSQL port) to query the PostgreSQL database which also resides on the Linux computer. Client software must query both the piano server software and the database in tandem to perform operations, control the piano, and receive updates from the piano. This information is summarized in Fig. 3. We shall hereafter refer to the piano's server software as the "server," and any connected device (PRC, DLRC, etc) as a "client."

The database stores a plethora of information including attributes of songs stored in the hard drive, USB drive (if inserted), CD-ROM (if inserted), floppy disk (if inserted), playlists, locations of backgrounds, saved preferences, and more. Please see the Database Details section for more information.

FIRMWARE CHANGES

We originally started this project in the summer of 2006 using a Disklavier running the v1.0 firmware. When the v1.2 firmware was released in September 2006, some parts of the messaging protocol changed, breaking compatibility with our software. Nevertheless, because these changes were an improvement over the previous firmware, we welcomed them, even though they increased the complexity of the protocol.

First of all, we noticed that the new firmware for the PRC uses the OFFSET and LIMIT keywords in their SQL queries to only request information that can fit on the screen at any one time. This reduces delay in the user interface, which could result from requesting a large number of entities (instruments, songs, etc) at once. Additionally, the v1.2 firmware regards case more strictly. For example, we had previously been accessing a database table using "USB_song", but with v1.2, only "usb_song" was recognized.

Another welcome change in v1.2 is one which optimizes network bandwidth by reducing unnecessary traffic. Previously, the piano server would send its many state variables to the controllers upon connecting. The server would also send the updated parameter (for example, quiet mode on) to all connected clients when a parameter was changed, even to the requesting client. With v1.2, however, the server sends nothing unless specifically requested to do so, allowing a device to only query the information it is interested in during connection set-up. Furthermore, instead of sending an updated parameter (such as volume, quiet status, etc) back to the client who requested the change, the server only updates the other clients. Thus, the requesting client is responsible for remembering the state it just requested. While this scheme reduces network traffic, it is less reliable because there is no confirmation of the state change to the requesting client. The request could get lost in transit and the client would think the server is in the recently requested state, when in reality, it is not. Besides this disadvantage, unpredictable results occur if two clients

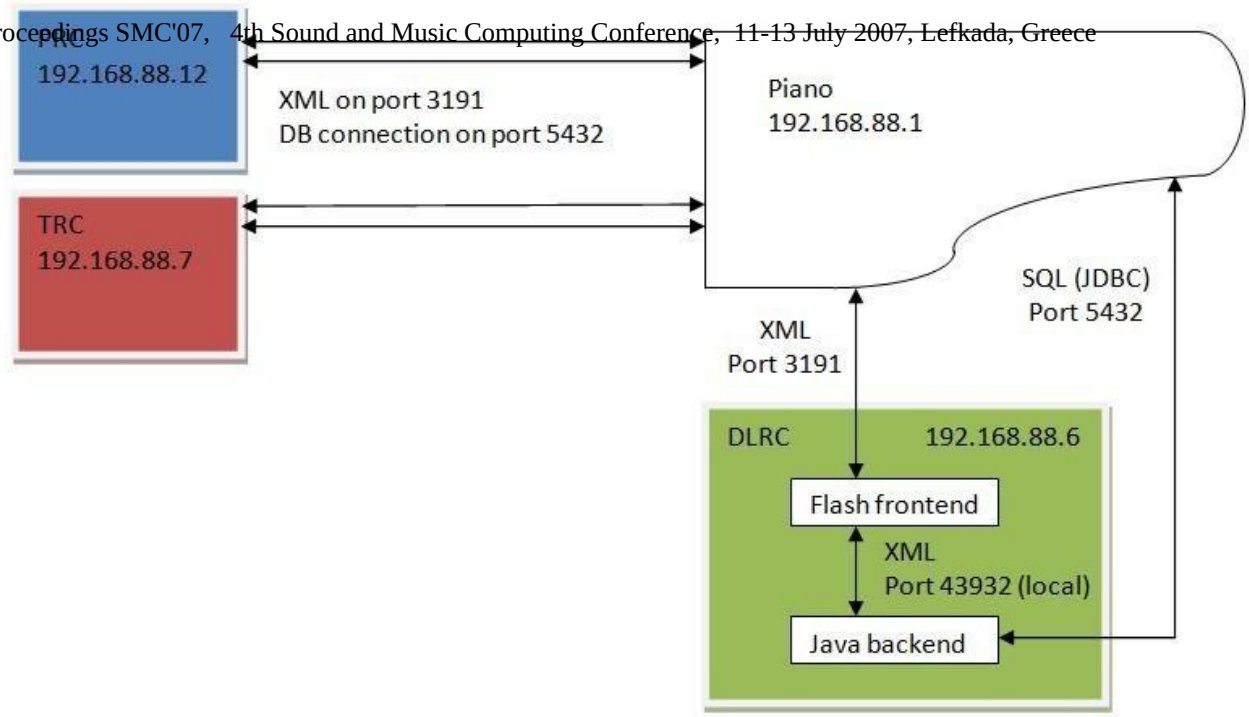

Figure 3: Architecture of the Disklavier Mark IV system, along with our own software architecture.

request the same parameter to change to two different values simultaneously, since each client will think the piano is in a different state. Yet because the piano operates on a fairly-reliable WLAN, these problematic situations should not arise very frequently.

Without a doubt, Yamaha will continue to improve the Disklavier server and client software, continuing to add new functionality and features. We plan to let the open source community continue to adapt the DLRC software to future firmware changes, as well as adding new functionality. We will gladly provide assistance to parties interested in improving our software. Our software and source-code is available at the authors' webpage: http://music.arts.uci.edu/dobrian/disklavier

TABLE I.
COMMANDS CLIENTS CAN SEND TO REQUEST SERVER STATE INFORMATION

Request	Description
<sync />	Used to synchronize the audio stream with MIDI data
<track_param />	Gets track parameters from server (tracks 1-16, new in V1.2)
<metronome />	Get metronome information, such as meter and tempo, from server (new in v1.2)
<autoplay />	Determine whether autoplay is on or off
<ab_repeat />	Get the current song repeat settings (new in v1.2)
<playfunc_status />	Get information related to song playback, including milliseconds into current song
<quiet_status />	Determine if the piano is in acoustic mode or quiet mode
<smartkey_prompt />	Sent just before a song starts playing and piano is in player mode
<panel_voice />	Gets information about the synthesized voice selected (quiet mode only)
<master_tune />	Gets piano tuning information (electronic tuning only)
<performance />	Get performance information (selected instrument, LH on/off, RH on/off etc)
<cdsync />	Used to synchronize playback with a CD.
<vocal_harmony />	See if vocal harmony is enabled or disabled
<play_back />	Get information related to playback such as time in song, volume, etc

METHODOLOGY

Our first goal in this project was to determine the way the Disklavier server software communicated with the PRC and TRC. We used the Ethereal program to obtain the raw packets which were being sent wirelessly from the controllers to the piano server. The server process on the piano's Linux computer communicates via XML messages. In order to determine which XML messages caused particular functionality, we would perform some action on the PRC, take a short packet capture, and then peruse the packet stream until we found the messages that resulted in the functionality we were looking for.

We first started with perhaps the simplest functionality: that of switching between quiet and acoustic mode. All that is necessary to invoke this change is to send the following piece of XML: `<quiet_status mode="acoustic"/>` or `<quiet_status mode="quiet"/>`. Conveniently, Flash has a built-in class called XMLSocket which allows XML to be sent and received using Actionscript. This fact influenced our decision to write the software in Flash. Additionally, we choose Flash because we surmised that the original software was written in Flash, seeing that it made use of transparency and animation. Our hope is that the DLRC interface can be made more similar to the interfaces of the existing solutions, in order to make the new software even easier to use.

The server software alerts clients to its current state by broadcasting XML messages periodically. The most common message contains pertinent playback information and is sent every second using the sequencer status message: `<seq_status status="stop" source_id="user" album_id="1" sel_song_no="4" song_id="4" time="0" sync="off" mode="master"></seq_status>`

This tells the client where the currently-selected song is located (user library, usb, cd, floppy, etc.), the album and song number (the database stores the song names and other information), milliseconds into the song, and other information. For more details on the various commands

sent between clients and the server, please see the Command Structure section.

One challenge we faced when developing our software was accessing the database synchronously. As far as we could tell, the original software used synchronous database queries; that is, the client software halted computation until database queries returned. This approach makes the program's code structured and organized. The XMLSocket class, however, only supports asynchronous communication. Since we were using this class to communicate with the database, we had to write our event handlers somewhat awkwardly. This complication was compounded by the fact that Flash cannot send SQL queries directly. There are three solutions to this problem [5], and we choose the least-intrusive option in order to avoid changing the piano's server software. Thus, we created a backend Java program which speaks XML to the Flash frontend, and uses JDBC to connect to the database. There do seem to be ways to communicate with a database synchronously using Flash, but such methods required expensive third party software. Since we wanted to release our program into the open-source domain, we decided to go with the free alternative. Perhaps this part of the software can be improved in the future.

DATABASE DESCRIPTION

The piano's computer runs a PostgreSQL database server (version 7.3.9). Multiple databases can exist on one PostgreSQL database server. In the case of the piano server software, there is the main database called "mk4db," (see Fig. 6) as well as an unused database called "garbage." This database appears to be an initial prototype that was abandoned. One table in this database is called "gaku"; this is the only hint of Japanese influence in the software. The large amount of English used in the rest of

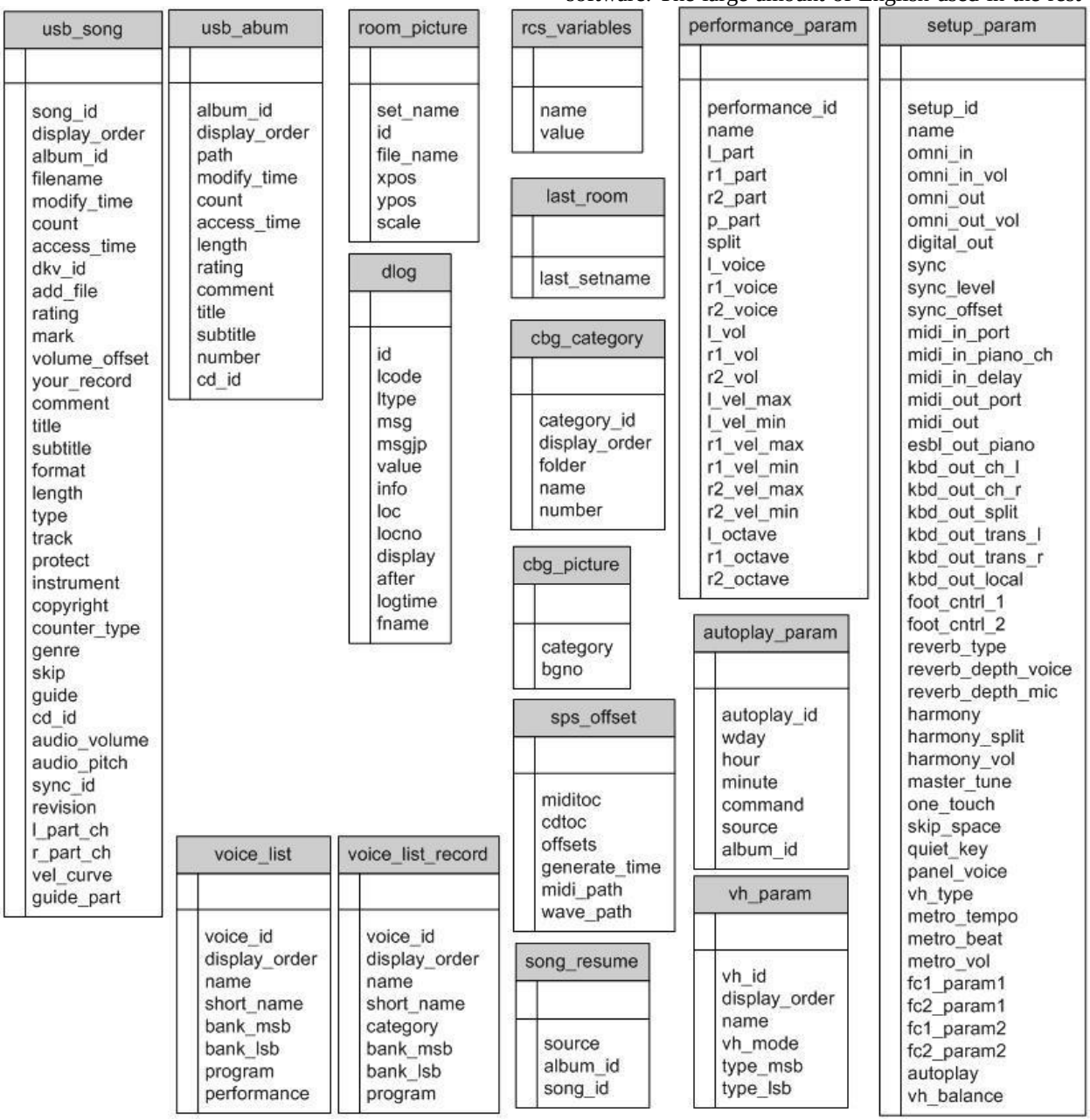

Figure 4: A summarized database schema. Other song and album tables of the form x_song and x_album are omitted because they have the same format as usb_song and usb_album.

the messages leads us to believe it was programmed by native English speakers.

The database stores a plethora of information (see Fig. 4). Most tables store song and album information. Each storage medium has two tables in the database: one containing the albums on the medium, called x_album, and the other containing all the songs with their attributes, called x_song (where x could be usb, cd, or fd). In addition, each built-in song repository, such as the Pianosoft library, Pianosoft Plus Audio library, user-recorded songs, playlists, MIDI radio songs, diagnostic songs, and demo songs have tables in the same format—for example, pianosoft_album and pianosoft_song or demo_album and demo_song. The server software populates the appropriate tables when a particular media is inserted into the piano. As an interesting aside, even though the piano has 2 USB ports, users can only access songs on the first USB drive inserted if two drives are plugged in at once. This may be fixed, of course, in a future firmware release (2.0).

Additional information besides song and album data resides in the database. Many tables are dedicated to storing parameters and settings. The table voice_list stores all the electronic voices that the instrument can play when in quiet mode. Other tables control art preferences for the TRC and PRC. Of particular interest is the dlog table, which stores logs and error messages that are not accessible to users via the PRC or TRC. The software makes an entry each time it is turned on, updated, tested, or notices something is wrong (such as a low power supply voltage). While the DLRC cannot display this information yet, it would certainly be a useful and informative feature to help end-users diagnose problems with their piano.

The database does not designate any columns as primary keys; rather, PostgreSQL's OIDs feature is used to assign a unique, six-digit integer to each entry in the entire database. This Object-ID takes the place of a primary key. Overall, the database is much more complex than one would initially expect.

SERVER SOFTWARE RESILIENCE

We were impressed with the server software's elegant handling of erroneous input. If a client sends a command to the piano that it does not recognize, it will return a XML NAK message: `<status name="some_command"><reply>NAK</reply><reason>unknown command.</reason> </status>`. If, on the other hand, a valid XML message is sent, but with erroneous parameters, different behavior can occur. For instance, the PRC allows users to vary the tempo from 50% to 120% of the original tempo. When we sent our own messages, we were able to vary the speed from 0% (stopped) to 120%. If values outside of this range were input, the software would limit [i.e., clip] the input to the nearest valid number. We also tried pitch transposition. The original software allows transposition between -24 and 24 semitones. We found that the server software checks incoming values and disallows values outside of this range. The addition of server-side input checking reflects good software design.

COMMAND STRUCTURE

We present a small subset of the messaging protocol used between the Disklavier and its wireless clients. These messages are some of the most common and widely-used commands which should be of interest to prospective Disklavier developers. Note that this protocol, the database schema, and other parts of Yamaha's software are subject to change without notice.

`<active />` Clients send this every three seconds so the server knows whether or not clients are still in range. The server also sends this message back to clients. If the server does not receive any XML messages from a client for 15 seconds, the server closes that connection, so sending this message avoids potential disconnects.

`<rcs_status status="on"/>` A client can send this message to tell the remote control server to come in or out of standby. When in standby, clients communicate with the server less frequently and the piano's computer is in a low-power state.

`<seq_status status="stop" source_id="user" album_id="1" sel_song_no="3" song_id="3" time="0" sync="off" mode="master" mtc_mode="off"></seq_status>` This message is sent by the server every second to update clients with information about the current playing song. Status can be stop, loading, loaded, play, or pause. Source_id is the location of the song – in this example, the user library. Sel_song_no references the song's display_order number, which is different from it's song_id number. Time is the number of milliseconds the piano has played through the song.

`<vol_status main_acoustic="100" main_quiet="7" main_headphone="50" voice="13" tg="33" audio="18" mic="10" />` This command informs clients of current volume levels and can also be sent by clients to invoke changes in volume.

`<playfunc_status trans="0" repeat="off" tempo="100" right_hand="on" left_hand="on" pedal="on" accompaniment="on" />` This message contains transposition amount, repeat settings, and tempo. It also lets the client know which parts of the piano will be robotically controlled. Like other commands, this message can be used as a status update or a way of invoking changes in these parameters by clients. Note that these parameters cannot be changed individually. Instead, the entire message (containing all attributes) must be sent in order for the server to recognize the message.

`<panel_voice_harmony harmony = "1"/>` Used to enable or disable electronic voice harmony.

`<master_tune mode ="0" master_tune="0"/>` States/changes the current tuning adjustments of the electronic piano used in quiet mode.

`<message_box />` Sent by the server with additional information to cause a client to pop up a message box to inform the user of some event. For example, it is used when deleting a song.

`<performance mode="0" split="28" p_part="1" r1_part="1" r1_voice="34" r1_vol="100" r2_part="0" r2_voice="18" r2_vol="100" l_part="0" l_voice="23" l_vol="100" />` This is sent when the user manipulates the piano's performance parameters. Different electronic voices can be used for the upper and lower parts of the piano, each with different volumes.

`<cdsync status="off"/>` Enables or disables syncing with audio CDs.

`<vocal_harmony vh="off" vh_type="3"/>` Enables or disables vocal harmony when a microphone is attached.

`<play_back skip_space="1" quiet_key="1"/>` Determines whether or not the keys are mechanically depressed and blank space at the beginning or end of songs is skipped.

`<smartkey_prompt key="" visible="off"/>` Can make the piano slightly depress keys for use in particular playback modes.

`<load_song source_id="pianosoft" album_id="6" sel_song_no="46"/>` This is sent by a client to make the piano load a particular song. The client must provide the database table where the song resides (source_id), the album, and song number.

There are many other commands sent from the piano that are not included here. Since we were interested in replicating the core functionality of the piano, we did not take the time to study how each command works. Nevertheless, we are impressed with the complexity of the Disklavier software and applaud the developers for using standard, easy-to-parse XML.

SECURITY

Throughout our survey of the Disklavier's inner workings, certain parts of the software concerned us with regards to security. Of highest concern is that the database has no password. Anyone within wireless range of the piano can log in to the database with a fairly unimaginative user name and a blank password. Once someone has access to the database, they could wreak havoc, possibly causing the client software to crash or behave in an unpredictable fashion. The location of art assets could be deleted, songs could be removed from playlists, etc. This also means that an attacker could change the protect flag on a file to remove that song's copy protection mechanism. We have not tried any destructive actions for fear of damaging our Disklavier since we wouldn't want to send it back for a firmware reinstall. We present the above information in hopes that Yamaha will be aware of these possible problems. In no way do we advocate using this information for nefarious purposes. Most likely, these decisions were made because the designers assumed that users of the piano would never discover how to access the database. With a growing population of technically-skilled musicians, however, we argue that the lack of database security is cause for concern. Adding a password to the account, even one that could be sniffed wirelessly, would deter all but the most intrepid hackers.

CONCLUSION

Throughout the process of interfacing with the Disklavier Mark IV piano, we learned much about the complexity of the piano's software. We were able to understand the messaging protocol used between the piano software and its original wireless controllers. Using this information, we developed the first third-party software controller for the piano, adding new functionality while allowing a convenient new usage paradigm to be implemented. We hope that others will use the information contained within this paper to come up with creative new ways of controlling the Disklavier, perhaps from other devices (Java-enabled cell phone?). In addition, we hope that manufacturers and entrepreneurs will use a similar protocol to control other robotic instruments which are currently under development, in order to promote an open standard for musical instrument control.

REFERENCES

[1] "Yamaha Player Piano Goes Wireless: The Disklavier Mark IV" http://www.prnewswire.com/mnr/yamaha/20680/.

[2] Ethereal: A Network Protocol Analyzer. www.ethereal.com

[3] Yamaha Corporation. "Disklavier Technology Press Release - November 18, 2004", http://www.prnewswire.com/mnr/yamaha/20680/docs/Mark_IV_Technology_Release-FINAL.doc.

[4] Dayson, Phil. "Wireless Control of a MIDI Player PC." http://mmd.foxtail.com/Archives/Digests/200502/2005.02.13.03.html.

[5] Petrescu, Razvan. "Connect Flash to a Database Using Sockets", http://www.devx.com/webdev/Article/30638