Software Requirements Specification

for

MIDI Humanizer

Version 1.0

Prepared by Jacob Peck,
Jason Shaffner,
and Jamie White

State University of New York
College at Oswego

September, 2011

Table of Contents

1. Introduction

1.1 Purpose

MIDI Humanizer exists to fill a niche in the MIDI representation of performance aspects of a musical piece. The program reads in a MIDI file specified by the user, and applies profiled variance to various dimensions of MIDI voices, such as volume and timing. MIDI Humanizer attempts, in a few words, to humanize a MIDI sequence.

1.2 Document Conventions

When reading this document, it can be assumed that any `monospaced` characters are either direct input from a user, or output from an operation.

1.3 Intended Audience and Reading Suggestions

This document is prepared for developers, project managers, users, and documentation writers. Those who wish to read this document are suggested to approach it in a linear order, after first glancing at the section headings as outlined in the table of contents. Of particular note is section 3, describing all of the vital requirements from a developer's perspective.

1.4 Project Scope

MIDI Humanizer is an attempt to reintroduce the human element to MIDI streams, particularly those that have been quantized. This will allow computer musicians to simulate human imperfections. This project seeks to offer computer and MIDI musicians an option to increase variance in a simplistic way. In terms of marketability, the potential for this product is low in the general market, but rather high in the digital audio workstation market.

1.5 References

This document refers to the General MIDI specification, which is available online at <http://www.midi.org/techspecs/index.php>, as well as the JFugue User's Guide, which is available for purchase for $25 from <http://www.jfugue.org/>. The initial project description provided by the client is also referred to, along with any references that the client may provide.

2. Overall Description

2.1 Product Perspective

MIDI Humanizer is a stand-alone project operating in a field with little competition. MIDI Humanizer does not seek to replace any available projects, nor does it seek to add on to any existing products.

2.2 Product Features

REQ22-1: MIDI Humanizer will offer a simple graphical user interface.
REQ22-2: The interface will allow the user to import files to be worked upon, select variable levels of variance and channels to operate on, provide an output file, and initiate the process.
REQ22-3: MIDI Humanizer will allow for previews during every step of the way.
REQ22-4: The variance applied to the sequence will be determined by a user-selectable (and user-definable) profile, allowing for emulating different styles of playing.

2.3 User Classes and Characteristics

Computer Musicians and Mixing Technicians – These users have a decent level of familiarity with MIDI, and as such should find no problem using this software. Advanced options such as profiling will be geared towards these users.
Amateur Musicians – These users may not know much about the MIDI spec, but they are able to click buttons and receive results. Advanced options will be hidden from them if they choose.
MIDI researchers – These users know the MIDI spec inside and out, and may find that this program doesn't suit their needs.

2.4 Operating Environment

REQ24-1: MIDI Humanizer will be written in Java, and will run anywhere where there is a JDK available that implements a standards-compliant MIDI device. OpenJDK will work on several platforms (excluding ARM).
REQ24-2: The preferred platform will be the official OracleJDK, which has the widest coverage of standards-compliant MIDI device implementation. Any version of the Oracle JDK from Java 6 and beyond will work, though preferential treatment will be given towards the optimizations present in Java 7.

2.5 Design and Implementation Constraints

REQ25-1: Source code will be optimized for maintainability and reusability.
REQ25-2: Source will be documented with comments (make "bus code").
REQ25-3: The project will stick solely to the MIDI interface.
REQ25-4: All programmers will be expected to use javadoc to document their methods and classes.

2.6 User Documentation

REQ26-1: On-line help will be available within the program at the user's request.
REQ26-2: The interface will be intuitive.
REQ26-3: Help will be developed as the GUI develops.

2.7 Assumptions and Dependencies

This project assumes that MIDI and JFugue will work as advertised.

3. System Features

3.1 Input/Output

3.1.1 Description and Priority

This feature has a high priority. This feature will read in and send back out a MIDI file. The benefit of having a I/O stream is that it can read in most kinds of files already we just need to do minor adjustments so it can read in a MIDI file. Penalties of this is that we have to find a way to make it platform independent so it could work not only on Windows but on Mac OS and Linux as well.

3.1.2 Stimulus/Response Sequences

The user the user will request a file be loaded. This will be handled by the file I/O method. A message will show up confirming that the file was received and it was in the right format or that it was not received or it was a null file. A pointer will be given no how to fix the mistake but ultimately it is up to the user.

3.1.3 Functional Requirements

REQ31-1: In Order to use this feature this user must have a MIDI file as a input and java must be on the machine. The method will respond to reading in the file and use a series of exceptions to catch a error in the file if the file is of poor quality and tell you the length of the file.
REQ31-2: Should the user try to send in a invalid file type the user should get a message telling them that the file type was invalid and give them a clue to the proper extension.
REQ31-3: This Function should read in the file to be edited and Send back the file when this is complete. This method will also try to catch errors in the file and give the user a warning when a error is found and a easy way to fix it.

3.2 Algorithmic Manipulation of Random Input Sources

3.2.1 Description and Priority

This feature has a high priority. The algorithms must analyze the MIDI sequence, and manipulate them to provide a level of randomness to both amplitude and timing of individual note events. The randomness should move along a bell curve to provide most-probable simulation of said characteristics, while normalizing subsequent notes, as to provide consistency in tempo.

3.2.2 Stimulus/Response Sequences

The user will be able to select “humanness” profiles at various levels, such as “beginner” which will simulate a radical degree of imperfection (along a wider bell curve), or “expert” which will allow for finer imperfections (along a narrower bell curve).

3.2.3 Functional Requirements

REQ-32-1: The algorithms should be applicable to a wide range of inputs, such as different instrumentations, multiple channels, different tempos, etc.
REQ-32-2: The Box Cox Transformation will be applied to create random numbers over a standard normal distribution rather than a uniform distribution.

REQ-32-3: A Gaussian algorithm will be provided in addition to the Box Cox Transform.

3.3 MIDI Sequence Manipulation

3.3.1 Description and Priority

This feature has a high priority, and is critical for the delivery of a finished product. This module of the software will comprise taking information provided by the above two modules and manipulating it so as to add a more human "feel" to it, before sending it back out to the I/O module for writes to disk and sonification if desired.

3.3.2 Stimulus/Response Sequences

This action is performed upon the user confirming via the user-interface that humanization should take place. Humanization happens in the background, and is a non-interactive process, after determinations such as profile, parameters, and input have been determined and confirmed.

3.3.3 Functional Requirements

REQ-33-1: The process must be invisible to the user, to provide a sane user experience.
REQ-33-2: The process must have selectable parameters of the input stream and work upon them individually (i.e. volume, timing, etc.), to allow the user to apply humanization to only certain factors.
REQ-33-3: The process must supply the algorithmic random number generator with a user-selected profile curve.
REQ-33-4: The process must happen in memory and only be committed to disk upon user confirmation.
REQ-33-5: The process must be user-previewable before confirmation to save.
REQ-33-6: The I/O subsystem and algorithmic random number generator must be loosely coupled, to ease maintainability (i.e. use standard JDK and JFugue classes and encapsulate behavior).
REQ-33-7: The process must ensure that the length of the input and output are consistent lengths within a certain tolerance value.

4. External Interface Requirements

4.1 User Interfaces

REQ41-1: There will be a graphical user interface with a unified design.
REQ41-2: A tool bar at the top of the frame will serve as the main method of navigation.
REQ41-3: Dialog boxes will be used to confirm user actions.
REQ41-4: There will be a help option.
REQ41-5: Standard Swing components will be used to create the interface.

4.2 Hardware Interfaces

REQ42-1: Standard input/output devices (monitor, keyboard, mouse, speakers) will be used.

4.3 Software Interfaces

REQ43-1: MIDI Humanizer will make use of the Java platform, and also the JFugue library, version 4 or higher. The Java platform runs on the Java Virtual Machine (JVM), and is available on many platforms.
REQ43-2: The JVM will communicate with the Operating System in order to read in and write out files and communicate with hardware.
REQ43-3: JFugue will communicate with the objects presented by the JVM representing the data on disk. Such objects include representations of notes, sequences, and MIDI files themselves.

4.4 Communications Interfaces

REQ44-1: The only communications handled by MIDI Humanizer is producing sound through JFugue and writing files to disk through the JDK's interface with the OS.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

REQ51-1: The program should not "stall" for longer than 5 seconds when not performing a computation.
REQ51-2: Transformations upon MIDI sequences should not take longer than 5 seconds per minute of MIDI events.
REQ51-3: The interface should be lightweight, such that the controls do not get in the way of user interactions.
REQ51-4: During long periods of computation, a progress bar should be displayed.

5.2 Safety Requirements

<Specify those requirements that are concerned with possible loss, damage, or harm that could result from the use of the product. Define any safeguards or actions that must be taken, as well as actions that must be prevented. Refer to any external policies or regulations that state safety issues that affect the product's design or use. Define any safety certifications that must be satisfied.>

5.3 Security Requirements

There are no such concerns with this product.

5.4 Software Quality Attributes

REQ54-1: Should be cross-platform.
REQ54-2: Should be relatively light on system resources. 1Gb ram as a maximum (not due to software bloat, but to allow overhead for JDK), and around 1.2Ghz processing speed.
REQ54-3: Should confirm any "dangerous" user actions, such as file overwrites.
REQ54-4: Should preview MIDI sequences smoothly without hiccups.

6. Other Requirements

N/A.

Appendix A: Glossary

Bus Code: Code so well documented that you could be run over by a bus tomorrow and someone else would be able to pick up where you left off with no problem.

Humanization: The process of applying variance to a quantized MIDI sequence.

JFugue: A Java library for manipulating music. Includes several routines for manipulating MIDI data.

JDK: Java Developer Kit. A collection of Java classes and tools designed to ease Java development.

JVM: Java Virtual Machine. The artificial platform that comprises the core of Java technology.

MIDI: Musical Instrument Digital Interface. An industry standard protocol for the exchange and storage of musical data.

Profile curve: A probability distribution function (pdf) that models the probabilities of variance from some standard average (mean). A good example is the Standard Normal Distribution curve (also called the “bell curve”).

Quantized: A MIDI sequence that has been quantized has been crafted in such a way as to discourage variance, in that all note-on and note-off events are rounded to the nearest multiple of a duration, say the nearest quarter-note.