My Beats
Integration Plans

CSE 4904: Design Lab
Milestone IV
Michael Smith
Purpose and Scope:

The purpose of this document is to define the integration strategy that will be used to integrate the different modules of the MyBeats project. It will describe the general strategy used throughout for integration, and then specific integration details needed by the various modules. The entry conditions for each module that must be met prior to integration will be defined along with the exit conditions which mark successful integration. Between these points, the process of testing the integration, along with error resolution will also be detailed. Beyond the integration process this document will also detail the remaining work schedule and responsibilities for the next month.

Integration Strategy:

Entry Criteria:

The entry criteria define the base requirements that each module must meet before being integrated with the rest of the system. In order to be integrated each module must meet two main conditions. The first is that the module has been thoroughly unit tested. This is to ensure that on its own, the module will not cause problems that are more easily resolved while the modules are separated. The second main criteria is that each module has an entry and exit point. The need for these points is due to the fact that the MyBeats project is a start to finish process where the modules are the steps to be taken along the way. As such, the entry and exit points are the connection points between the different modules. The entry point must be able to accept all the necessary data for the module to function. The exit point must be able to yield the results of the module for use by the next module. The following diagram shows the order of the different modules that are involved in the MyBeats project. The exact entry and exit points will be defined for each module at a later point within this document.
Integration:
Upon integration, each module will be connected using the entry/exit points in the proper order. For the modules to be integrated, it must be assured that the data that is passed from one module to the other is packaged correctly for the modules to handle it. Once the modules have been composed together, than they can begin their integration testing. The integration testing will be different from the unit testing. The unit testing was to ensure that the module had no internal errors. Integration testing is to ensure that one module does not cause problems to another module that were unaccounted for prior to integration. As such, what this testing needs to ensure is the following. For each connection between the modules, each entry point must be able to account for all possible values yielded by the previous modules exit point. If this is the case, then the connections will not result in any errors and the system will be functional as the internals of each module will have already been tested. If this is not the case, and one module is able to pass results to another module that the second module cannot handle, than a bug will result as the module will not know what to do. If this is the case, the error resolution process defined below will be employed.

Exit Criteria:
Exit criteria defines the conditions necessary for the integration of the different modules to be determined successful and for integration to be concluded. When all possible results and inputs of each module have been accounted for, and handled correctly along the MyBeats process, the system will be considered successfully integrated. Most projects upon successful integration are ready to go to Beta testing as the internals and integration of each module will now have been tested and found to work. As the MyBeats project relies heavily on heuristic functions to produce the end result, after integration, the MyBeats project will be able to enter a tuning stage. This stage is similar to Beta testing in that people will use and evaluate the system. However, unlike Beta testing, this tuning stage will incorporate constant tweaks to the heuristic values and the internal algorithm that utilizes them in an effort to produce a better user experience.

Error Resolution Procedure:
When an error is found during the integration process, the following methods will be followed. First, integration must halt in order to resolve the current error. When an error is detected, it will be recorded within the source control system. This is for our book keeping purposes to track errors. Then the modules that have been integrated will be analyzed to determine where the error is being generated and at what link in the process the error is causing the modules to fail. At this point a rework must be created to account for the circumstances that are causing the error and then implemented. Upon implementation of a fix, the module must then be unit tested again to ensure that new internal errors have not been created. After the new round of unit testing the module can be connected to the system again. At this point the problem can be marked as resolved within our source control system and integration testing can resume.

Module Specific Integration:
When referring to the module flow chart found above, it is clear of the operation of the MyBeats process should proceed. As such, the exact entry and exit points of the various modules can easily be defined so that the modules are able to successfully work together.

MyBeats Module:

The MyBeats module is the module that the user interacts with. As such, it has no module specific entry point as it is responsible for internally collecting the data from the user and the system configuration files. However, the MyBeats module is the module responsible for the rest of the program control flow. Despite the flow chart showing the modules as one after the other, these other modules will all be called directly from the MyBeats module one after the other. This is so the error detection at runtime from one module can be reported and then the system can be halted without extra modules having to be run. What this means is the MyBeats module will have to start the first module (AnalyzeInteraction) and pass the correct information to it. At the conclusion of that modules functioning, either an error will be detected in which case MyBeats will halt and display the error to the user, or the reprocessed data will be collected and passed on the the next module.

DataStructures Module:

The DataStructures module plays an important role in the functioning of the MyBeats project. In and of itself the DataStructures module does not do anything functional, it is more of a set of different packages for the results of the other modules to be wrapped in allowing information to be passed between the modules. It is what allows the results of, for example, the TrackGenerator to be passed to the MidiGenerator. This is basically responsible for the data integration of the rest of the modules.

OggGenerator Module:

The OggGenerator module just needs an entry point to receive a song location. Then using this location it will attempt to convert that song. It must also be able to report errors to the calling module, MyBeats.

Remaining Modules:

The remaining three modules, the AnalyzeInteraction, TrackGenerator, and MidiGenerator modules all have very similar integration requirements. They each must be able to accept a package from the DataStructures module, and return either a new package from the DataStructures module or a descriptive error as to what failed. Specifically, the AnalyzeInteraction takes the initial song information and will yield a RawSongInfo class instance. The TrackGenerator will take as input the RawSongInfo and return a SongTracks class instance. This SongTracks instance will then be used by the MidiGenerator to create the remaining files needed for the new Frets on Fire song.

Program Stubs and Test Data:

While in development, various program stubs will be needed. As each module
can be developed semi-apart from the others, at any point where cross module interaction will occur before that module has been fully developed, a stub can be inserted to allow functional testing of the individual modules. The most notable occasion for this will be the TrackGenerator module which is the last module to fully be implemented and refined. The reason for this is in order to develop the heuristics and methods to create a track, the output needs to be easily available in order to judge the results. This is not possible before the rest of the system is up and running.

A couple of different places make use of test data during development. The first is the MIDIGenerator module. This module does not have to start off creating a good song, but can use randomly generated sequences of notes and timings in order to test its functionality and success with Frets of Fire. The second module needing special test data is the Track Generator module. As this module is responsible for generating ‘good’ playable song tracks, if simple song files are used that highlight different musical scenario’s, different methods can be developed to handle those scenarios one at a time. This is far advantageous to using a complicated song such as one off the radio. Using special songs to distill methods for various scenarios is far easier.

Responsibilities:

MyBeats Module -> Dylan and Karen
AnalyzeInteraction Module -> Michael
Track Generator Module -> Michael with help from Dylan and Karen
MidiGenerator Module -> Dylan
OggGenerator Module -> Karen
DataStructures Module -> Mike, Dylan, Karen
Unit Testing -> Mike, Dylan, Karen
Integration -> Mike, Dylan, Karen

Schedule:

Other than the TrackGenerator module, the rest of the modules are nearly completed and functional. The main snag is the MidiGenerator which is proving to be more difficult than anticipated to correctly format a file for Frets on Fire. The OggGenerator is also a larger issue than was initially thought though as a side process, it is not yet fully necessary for the integration and testing of the rest of the modules. Once the last bugs from the MidiGenerator are ironed out, a variety of different heuristics and algorithms that have been developed for the Track Generator can be tested and reworked until a decently working module is obtained. As mentioned above, the full implementation of the TrackGenerator module could not be done until the rest of the system was more or less fully functional. This is due to the nature of heuristics in that to evaluate and modify them, the end result needs to be evaluated. By the next presentation we should have a playable song track for demonstration, though not necessarily different difficulty levels as that requires constant development and adjustment of the heuristics.