

Final Deliverable

MCM

June 5, 2001

Jimar Garcia

Sal Ledezma

Cayci Suitt

Gene Wie

Table of Contents

I Integration Testing	3
II Unfinished Items	4
III Future Enhancements	4
IV Changes Made to the Requirements and Design Documents	5
V Performance	5
VI Appendix	
help.txt	6
complete.mcm	9
yoyoyo.mcm	10
MCM source code	11

I. Integration Testing

Data Sets and Testing Conditions:

Our team made extensive use of .v (pre-recorded motion capture) files and .mcm (mapping information) files to complete the integration testing of our system. The .mcm files used in testing were simple mappings of 3-6 rows of mapping data, usually named a derivative of *test*.mcm. The final testing .mcm file, complete.mcm, is included with this document. Another testing file, yoyoyo.mcm, was used to test things related to the entire body part listing. The one .v file we had to use, hvdemo07.v, included the motion of a person walking in a circle.

The .v file was used by the Vicon Real-time Emulator to feed a stream of data to the StreamReader module over the network port 800. The StreamReader module then passed the motion data to the Translator module, which then passed it on to the MIDIOutput module. The IO module, with the help of the MCMMMap module, created .mcm files, which were in turn read in by the MappingInfo module. The Translator module used the data structure created by the MappingInfo module.

The MCMMMap and MCMTranslate modules were tested separately before they were brought together on the same machine. The main MCMMMap module was integrated with the IO module by adding the information from the various GUI elements one at a time until the correct information on the whole was being written to the .mcm files. A similar yet reverse method was used to read the information from the .mcm file back into the GUI components for the file-open functionality. The interfaces between the various MCMTranslate modules were individually tested and integrated into the system as a whole. First, our team attached the StreamReader module to the Translate module assuring that data points could be read off the network and decoded via data structures. By running the StreamReader module and printing out the data points the connection between these two modules was verified. Next, the team included the MappingInfo module in the project and initialized the class with the test mapping (.mcm) file. Queries were made on the MappingInfo object containing data stored in the mapping file. Finally, these two connections were fused by obtaining data from the StreamReader object and using such info to make the proper queries on the MappingInfo object. This gave verification by creating a MIDI command that could be sent to the MIDI module.

We performed full integration tests by installing both executables on the same machine and using one program to call the other. We created a .mcm file using MCMMMap, saved it, and then called MCMTranslate. The Real-time Emulator needed to have been running a .v file at this point to simulate a motion capture. The StreamReader module also needed to wait for the MappingInfo module to create the data structure before it could look for the stream data. We tested each data set multiple times to make sure that the same MIDI output was created for the same .v file and .mcm file pair.

We need to either move the system to the client machine or acquire more sample .v files from the client in order to do testing of simple motions and mappings.

Possible Problems:

The possible problems that may occur once we move the system over to the client machine involve the individual programs running as stand alone programs outside of their respective development environments. The team has had trouble running MCMMMap.exe on a machine on which it was not already compiled and running in a development environment. We found that this was due to MCMMMap requiring the Java Runtime libraries to be present on the host machine to run properly.

Unforeseen problems may also occur once the system must work with the Vicon hardware as we have as yet only tested it using the Real-time Emulator software.

Due to limitations in the Vicon API, MCMTranslate must be recompiled once the IP address of the machine the stream is coming from is known. The IP address of this machine must be hard-coded into our system.

II Unfinished Items

Benchmark Speed Testing:

We have not been able to test if the speed of the camera system limits our ability to make MCM real-time. This was one of our risks, however we are currently confident that MCMTranslate operates fast enough to make the system real-time due to the efficient use of time and space. Algorithmically MCMTranslate runs in $O(n)$ time, yet we have not been able to test it under normal operating conditions.

Allowing Multiple MIDI Commands:

Due to implementation decisions MCM does not currently have the functionality to allow the user to associate multiple MIDI commands with a body part/axis pair. MCMMMap does not give the user an error when they attempt such an association, yet MCMTranslate does not recognize more than one command for each pair. The query to the MappingInfo object only expects one MIDI command to be returned for a given body part/axis pair. This lack of functionality can be simply added to the system given more implementation time, with little affect on the performance of the system as a whole.

III Future Enhancements

Our team suggests the following future enhancements for MCM:

- 1) Adding a scroll pane to the main window of MCMMMap to accommodate more rows of mapping information.
- 2) Asking the user if he/she wishes to save the current open file before MCMMMap exits.

- 3) Including a more in-depth help file.
- 4) Allowing for multiple capture subjects.
- 5) Allowing for user-defined data points (i.e. body parts or objects).
- 6) Allowing for the motion to trigger the playing of .wav music files.

IV Changes Made To the Requirements and Design Documents

Changes made to the Requirements document:

Made extensive modifications to Section 3.4 Domain Specific Rules.

The Use-Case Scenarios were moved to Section 3.7, after Section 3.5, which defines the components discussed in the Use-case scenarios.

Updated Section 6 Test Plan to include environmental requirements.

Added several terms to the glossary.

Updated the Section 9 Meeting Minutes to show the latest Requirements meetings.

The user is forced to use the .mcm file format by the system.

The input minimum and maximum must be within the range of 0 and 300 centimeters, and the .mcm file will store the value as millimeters. The output minimum and maximum must be within the range of 0 and 127 (whole integers) because that is the range for MIDI commands.

Changes made to the Design document:

MCMMMap::MapData was changed to RowData.

Interface between StreamReader and Translator Included a data structure that served as a transport for reconstruction points. Data structure is called ReconstructionFrame. This structure contains four vectors that are representative of the data point's labels and values.

MIDIOutput module has been compiled as a static library by the project name of ImprovMod.

V Performance

The primary decision that aided in MCM's use of time and space was both a design and an implementation issue. We decided that the system would perform better if it were split into two separate executables, MCMMMap and MCMTranslate, each with its own set of modules. Since MCMMMap was primarily a user interface it needed to be "pretty" and user-friendly. It was decided to implement it in Java with Swing components to aid the creation of a usable GUI, without a need for speed. MCMTranslate was implemented in C++ since it runs as a background program, unseen, yet it needs to be fast enough to output MIDI commands in real-time. It would have been difficult to make a usable, familiar GUI using C++, yet Java would not have been fast enough to meet the real-time requirement.