

# **MCM Design Document Revision 1.0**

**21 May 2001**

**Cayci Suitt  
Gene Wie  
Sal Ledezma  
Jimar Garcia**

# 1. Introduction

The purpose of this document is to describe the design of the Motion Capture Music (MCM) software system. It primarily describes the architectural style of the system, the modules that comprise the system, and how those modules interact with one another.

MCM is a two-stage design at the top level, is comprised of two separate executable programs, MCMMMap and MCMTranslate.

**1.1. MCMMMap** accepts data from the user in order to create a mapping of C3D motion data onto MIDI data. The data input program will consist mostly of a graphical user interface through which the user can input, save and/or load his translation choices.

**1.2. MCMTranslate** uses the mapping to perform the specified mapping in real-time. The translation program acquires the data from the network port that connects the Vicon RealTime Workstation and the translated visual output via reader code. This recorded data is what is translated from the streaming C3D motion data format to MIDI music.

## 2. Project Plan

The following continuation of the project plan from the Requirements has been slightly revised given the time frame and actual progress on the system.

5-6	Architectural Design	All
6	Produce Design Specifications	All
6	Develop capture/translation algorithms	All
6	Prepare Technical Presentation	Gene, Sal
6-7	Give Technical Presentation, Submit Report	All
7	Complete Design Specifications	All
7-8	Implement Mapping Functionality ?? User interface ?? Disk I/O	Cayci Cayci
7-8	Implement Translation Functionality ?? Vicon Motion Data Stream Reader ?? Translator Process ?? MIDI Output	Jimar All Gene
8	Test Mapping Functionality	Cayci
8	Test Translation ?? Unit Testing ?? Integration Testing	Gene, Jimar, Sal All
8-9	Integration Testing (MAP+TRANSLATE)	All
9	Usability Testing	All
9-10	Performance Analysis	All
10	Final Demonstration	All
11	Maintenance, Revision	All

Module	Developer	Coded?	Unit Tested?	Integration Tested?
Map (main)	Cayci	Yes	Yes	No
Map::GUI	Cayci	Yes	Yes	Yes
Map::IO	Cayci	Yes	Yes	Yes
Translate (main)	Gene/Sal	No	No	No
Translate::Vicon	Jimar	Yes	No	No
Translate::MIDI	Gene	Yes	No	No
Translate::Translator	J/G/S	No	No	No

### 3. Design Specification

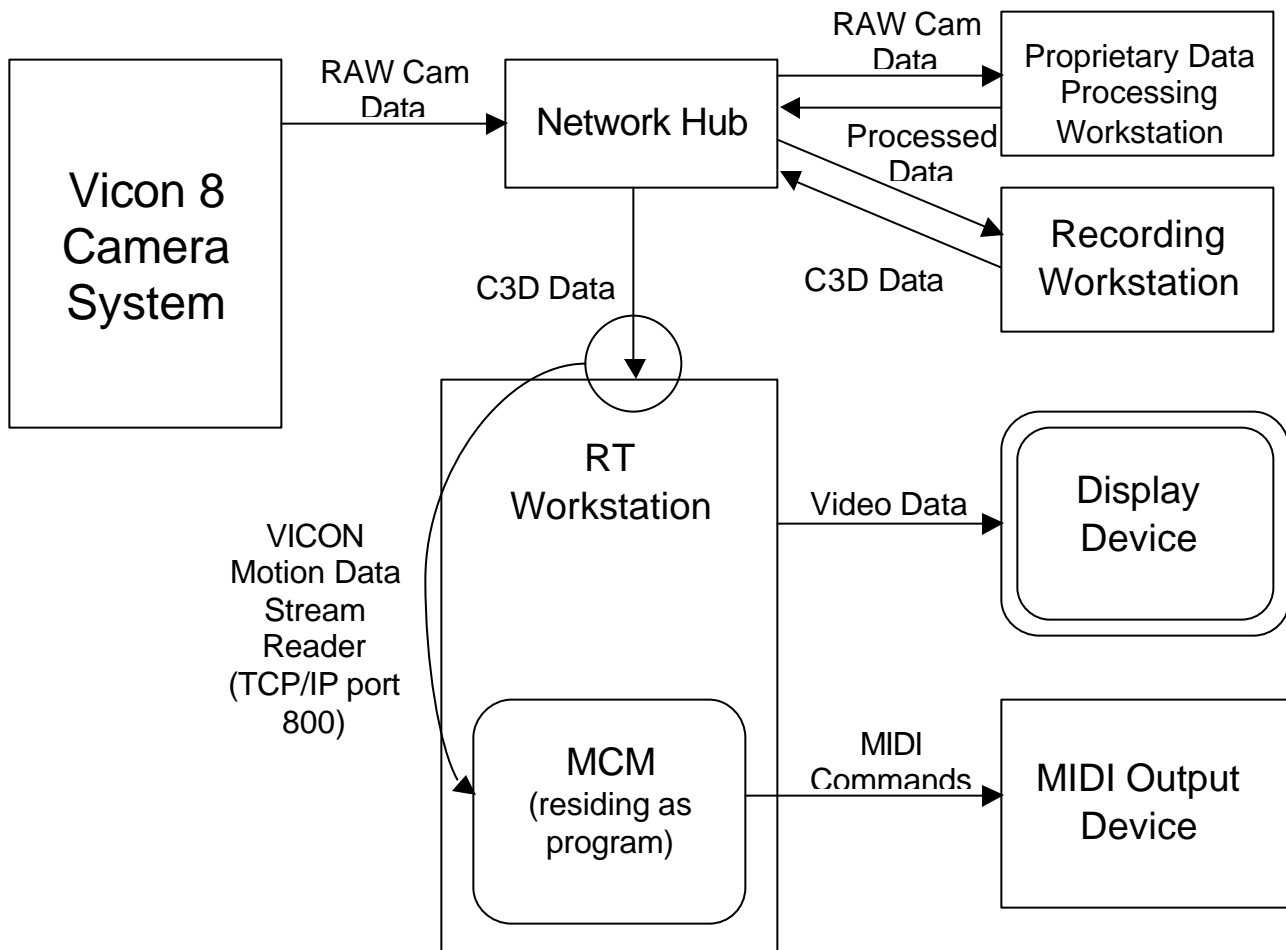
The overall MCM system has no specific architectural style; it is simply comprised of two separate programs, MCMMap and MCMTranslate.

MCMMMap could best be defined as a hierarchy of functions.

The architectural style for the Translation portion of MCM is Pipe and Filter. The MCMTranslate program/module adheres to this architectural style in that the data is collected, filtered, manipulated and then output again without any other program "knowing" the inner workings of this data translation.

The MCMTranslate needs the MCMMMap to function in that it cannot filter data without a mapping, but the reverse is not true. A user is able to create mappings without ever using them to translate 3D motion data into MIDI.

### MCM System Informal Data Flow



## **4. Subsystem Narrative**

### **4.1. User Interface**

The user interface of the Mapping portion of the MCM system is comprised of a GUI interface built in the Java programming language, that provides the user the ability to enter in information that correlates motion data to MIDI commands.

### **4.2. Input/Output (MAP)**

The Mapping portion of the MCM system, having received user input correlating motion data to MIDI commands, can save the specified “mapping” by the user to a file on disk.

### **4.3. Input/Output (TRANSLATE)**

The Translation portion of the system, which runs as a standalone executable, loads in the saved mapping on disk into memory.

### **4.4. Vicon Motion Data Stream Reader**

The Vicon-supplied motion data stream reading code reads the real-time motion data off the network; the data is sent in real time by the Vicon 8 camera system to its recording workstation and intercepted by this subsystem.

### **4.5. Translator**

The Translator Portion of the MCM system uses the mapping provided by the Map Portion of the MCM system to translate the motion data provided by the stream reader into corresponding MIDI commands. It is in here that the set of heuristics defined by the mapping is applied to the set of incoming motion data in order to transform it into an equivalent in sound data.

### **4.6. MIDI**

Improv 2.3.0 is a library of routines for MIDI command output that MCM uses to generate the sounds. The Translator sends the Improv-based MIDI module the relevant commands it has determined from the heuristics for the mapping to translation for immediate output of the sound to a General MIDI (GM) compatible device. Permission has been granted to use this code for the non-commercial purpose of academic research.

### **4.7. Limitations** on the current design include:

- 4.7.1** Full System for testing not present at UCI in complete form.
- 4.7.2** Dependency on external third-party code.

## 5. Description of Individual Modules

### MCMMap

**Purpose:** This is the top-level module that represents the data input portion of the MCM. This module will collect data from the user and save it to a file so that MCMTranslate can use it. The main() will exist in this module and will be the first program the user runs.

**In interface:** none

*// DEFINE: An "in" interface is what others call*

**Out interface:** none

*// DEFINE: An "out" interface is what [the module]  
// needs from others*

## 5.1 MCMMMap Modular Design

### MCMMMap::GUI

**Purpose:** This is the graphical user interface used for collecting mapping choices from the user.

**In interface:** none

**Out interface:**

```
String[][] getData(File mapname)
void SendData(String[][] data, File mapping, int rows)
void RowData.addDuration(String d)
```

### MCMMMap::IO

**Purpose:** This class handles IO operations for MCMMMap.

**In interface:**

```
String[][] getData(File mapname)
void SendData(String[][] data, File mapping, int rows)
void RowData.addDuration(String d)
```

**Out interface:** none

### MCMMMap::MapData

**Purpose:** This is the data structure format of the file that stores the user-specified mapping in the GUI.

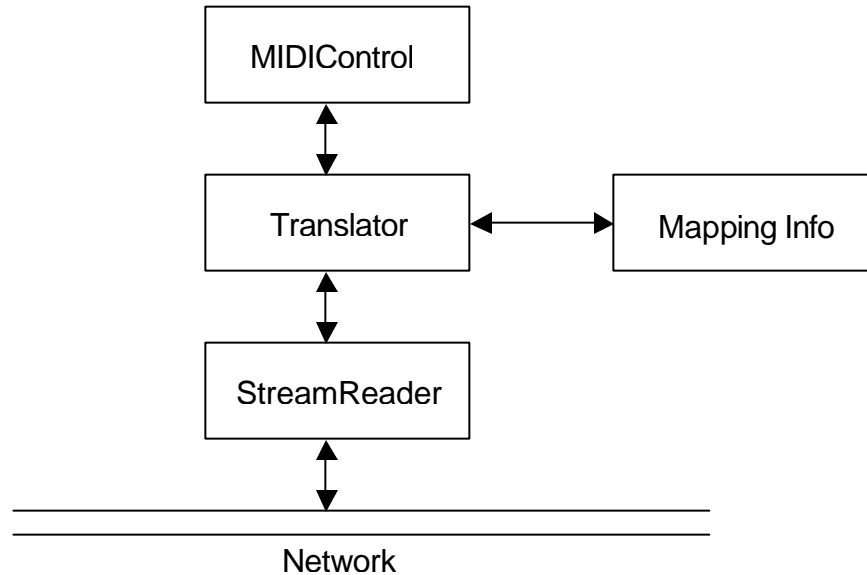
**In interface:**

```
void addDuration(String d)
```

**Out interface:** none

## 5.2 MCMTranslate Modular Design

5.2.1 MCMTranslate is comprised of four sub-components: Stream Reader, Translator, Mapping Info (I/O), and MIDI.



### 5.2.2 Interfaces

Module: MCMTranslate::StreamReader

Function Name	Return Type	Parameters	Description
StreamReader	None	None	Constructor
setIPAddress	void	char * <i>strIPAddy</i>	Sets internal variable that stores the IP address of the Vicon Real Time system
setHostName	void	char * <i>strHostName</i>	Sets the internal variable that stores the host name of the Vicon Real Time system
connect	bool	None	Connects with the Vicon Real Time system. This function will need threading so we can populate a local cache independently of being polled for reconstruction points.

cacheIsEmpty	bool	None	Returns TRUE if reconstruction point cache is empty.
getReconstructionPt	int *	None	Returns an integer array with three elements containing the x, y, and z coordinates of the queued reconstruction point.
isConnected	bool	None	Returns TRUE if still connected with the Vicon RT system.
queryDisconnect	void	None	Queries the StreamReader object that a disconnection request has been made. It will attempt to disconnect in the next network polling period.
getIPAddress	char *	None	Returns the stored IP address of the Vicon RT system.
getHostName	char *	None	Returns the stored host name of the Vicon RT system.

Module: MCMTranslate::MappingInfo

Function Name	Return Type	Parameters	Description
MappingInfo	None	None	Constructor
setMappingFilePath	bool	char * <i>strFilePath</i>	Sets the internal variable that stores the absolute path of the mapping file. This function returns a TRUE if the file exists, FALSE otherwise.
initialize	void	None	Initializes the data structures with all the mapping information from the specified mapping file.

Module: MCMTranslate::MIDI

Function Name	Return Type	Parameters	Description
			MIDI output for the MCM system is handled by Improv 2.3.0, a library of C++ classes primarily consisting of generalized MIDI I/O communication functions; the specific class MCM uses in the Improv 2.3.0 system for MIDI output is called <b>MidiOutput</b> and has the basic functions MCM will use to output the sound as follows (the use of this library allows MCM to directly send raw MIDI data to a General MIDI (GM) compatible device for immediate output (from Improv 2.3.0 MidiOutput Class Definition):

```
int cont(int channel, int controller, int data);
    Sends the value for the MIDI continuous controller command (0xb0) on the specified MIDI channel. Here is a list of continuous controllers.
```

?? *channel*: MIDI channel [0..15]

?? *controller*: MIDI continuous controller number [0..127]

?? *value*: MIDI data [0..127]

```
int off(int channel, int keynum, int releaseVelocity);
    Sends a note off command using midi command 0x80 with the specified key number and release velocity. Note that the most common way of turning off a note is to send the note-on command (0x90) with an attack velocity of 0 (see the play function.
```

?? *channel*: MIDI channel [0..15]

?? *keyno*: MIDI key number [0..127] (middle C = 60)

?? *value*: release velocity [0..127]

```
int pc(int channel, int timbre);
    Sends a MIDI patch change which changes the timbre on the specified channel. If you want to change to a timbre greater than 127, then check how your synthesizer does this. Usually, timbres are organized into banks, and you specify first which bank with the continuous controller #0.
```

```
int play(int channel, int keynum, int velocity);
    Sends a note on or note off on the specified channel. If velocity parameter is missing, then a note off command is sent.
```

?? *channel*: MIDI channel [0..15]

?? *keyno*: MIDI key number [0..127] (middle C = 60)

?? *value*: attack velocity [0..127] (note off = 0)

```
int pw(int channel, int mostByte, int leastByte);
```

Sends pitch wheel information on the specified MIDI *channel*. For the three parameter version, the *mostByte* = coarse tuning values 0..127 which are the most significant 7 bits of a tuning value, and the fine tuning values (*leastByte*) 0..127 are the least significant 7 bits of a tuning value. For the two parameter version, *tuningData* is a 14 bit number.

```
int pw(int channel, int tuningData);
```

Sends pitch wheel information on the specified MIDI *channel*. For the three parameter version, the *mostByte* = coarse tuning values 0..127 which are the most significant 7 bits of a tuning value, and the fine tuning values (*leastByte*) 0..127 are the least significant 7 bits of a tuning value. For the two parameter version, *tuningData* is a 14 bit number.

```
int pw(int channel, double tuningData);
```

Converts a number in the range from -1.0 to +1.0 into a 14 byte number which is sent out with the pitch wheel command.

```
void recordStart(char *filename, int format);
```

Starts recording MIDI output to the file *filename* according to the specified *format* which defaults to *ascii*. MIDI output sent through the *send* command are recorded, which output sent through the *rawsend* command are not recorded. If the file already exists, the file will be overwritten. The defined formats are:

- ?? 0 *ascii* format
- ?? 1 binary format
- ?? 2 Standard MIDI file format, type 0

Here is a [description of the formats](#) for recording MIDI output.

```
void recordStop(void);
```

Stops recording MIDI output to the file specified with *recordStart*.

```
void reset(void);
```

sends the MIDI command 0xFF which should force the MIDI devices on the other side of the MIDI cable (which is connected to the port of the *MidiOutput* object) into their power-on reset condition, clear running status, turn off any sounding notes, set Local Control on, and otherwise clean up the state of things.

int **send**(int command, int p1, int p2);  
Sends MIDI data from the computer to a synthesizer. If not recording, then just calls *rawsend* . If recording, bytes are stored in an output buffer until the buffer is supposed to be flushed. Then the time since the previous flush is calculated and recorded, as well as all the bytes in the output buffer.

?? *byte*: an 8-bit MIDI value to be sent out.

?? *flush*: 0=store *byte* in output buffer, 1=send output buffer data to MIDI I/O as well as current byte.

int **send**(int command, int p1);  
Sends MIDI data from the computer to a synthesizer. If not recording, then just calls *rawsend* . If recording, bytes are stored in an output buffer until the buffer is supposed to be flushed. Then the time since the previous flush is calculated and recorded, as well as all the bytes in the output buffer.

?? *byte*: an 8-bit MIDI value to be sent out.

?? *flush*: 0=store *byte* in output buffer, 1=send output buffer data to MIDI I/O as well as current byte.

int **send**(int command);  
Sends MIDI data from the computer to a synthesizer. If not recording, then just calls *rawsend* . If recording, bytes are stored in an output buffer until the buffer is supposed to be flushed. Then the time since the previous flush is calculated and recorded, as well as all the bytes in the output buffer.

?? *byte*: an 8-bit MIDI value to be sent out.

?? *flush*: 0=store *byte* in output buffer, 1=send output buffer data to MIDI I/O as well as current byte.

void **silence**(int aChannel = -1);  
silence MIDI data from the computer to a synthesizer. If channel == -1, then send note off commands on all channels, otherwise send only on specified channel.

void **sustain**(int channel, int status);  
Turns on/off the continuous controller 0x40 (sustain). Turn on sustain with *sustain*(1). Turn off sustain with *sustain*(0). Same as *off=cont(channel, 0x40, 0)*, or *on=cont(channel, 0x40, 127)*.

?? *channel*: MIDI channel [0..15]

?? *status*: on/off switch, [0..1] 0=off, 1=on

Module: MCMTranslate::Translator

Function Name	Return Type	Parameters	Description
Translator	None	None	Constructor
setRTHostName	void	char * <i>strHostName</i>	Sets the internal variable for the host name of the Vicon RT system. This variable will be used to initialize the StreamReader object.
setRTIPAddress	void	char * <i>strIPAdy</i>	Sets the internal variable for the IP address of the Vicon RT system. This variable will be used to initialize the StreamReader object.
setMappingFile	void	char * <i>strFilePath</i>	Sets the internal variable for the absolute path for the mapping information. This variable will be used to initialize the MappingInfo object.
initialize	bool	None	Initializes itself and all the objects it uses. It will initialize the StreamReader, MappingInfo object, and the MIDIController. This function will return TRUE if successful, FALSE otherwise.
run	int	None	Runs the translator, along with all subservient objects (ie. StreamReader). It returns an integer that represents a return code. A return code of 0 means that the function exited gracefully.

## 6. Integration Test Plan

The ITP for the MCM system involves a separation of testing procedures to encompass only individual modules first for each part (Map and Translate), followed by integration of single additional modules to build each separate portion to completeness.

<b>Test Name:</b>	Map Program
<b>Purpose of Test:</b>	To verify proper running of MCMMMap
<b>Module Interactions:</b>	Map::GUI, Map::IO
<b>Input Specification:</b>	Start program
<b>Output Specification:</b>	Program loads
<b>Test Environment Restrictions:</b>	Does not need MCMTranslate present

This is the most simple of the testing procedures, and is provided to ensure that the method for the user to create mappings is available.

<b>Test Name:</b>	GUI
<b>Purpose of Test:</b>	To verify functionality of GUI comment of MCMMMap
<b>Module Interactions:</b>	Map::GUI
<b>Input Specification:</b>	User manipulation of all modifiable graphical elements and text fields.
<b>Output Specification:</b>	No error messages upon data entry
<b>Test Environment Restrictions:</b>	none

There are numerous individual GUI elements in the user interface available. This includes the text fields used for range entry as well as the drop down menus that describe the available MIDI commands. Limitations on range as described in the requirements are implemented in the design and will feature in several iterations of this particular test.

<b>Test Name:</b>	Map IO
<b>Purpose of Test:</b>	To verify functionality of IO component of MCMMMap
<b>Module Interactions:</b>	Map::IO
<b>Input Specification:</b>	Save/Load file
<b>Output Specification:</b>	Save/Load file (corresponding)
<b>Test Environment Restrictions:</b>	none

The Input/Output routines must write a properly delimited text file of the mapping information to disk.

<b>Test Name:</b>	Translate Program
<b>Purpose of Test:</b>	To verify proper running of MCMTranslate
<b>Module Interactions:</b>	Translate::IO/MIDI/StreamReader/Translator
<b>Input Specification:</b>	Start program during capture
<b>Output Specification:</b>	Program executes during capture, generates sound
<b>Test Environment Restrictions:</b>	Can only be performed upon the completed testing and verification of functionality of the individual modules comprising this portion of the entire system.

This test should be the final step in integration testing. The mapping element of MCM exists as its own independent component, able to be used without the presence of this portion. However, this particular test is the complete run of all the significant system components that allows the motion capture system users to generate sounds in real-time.

<b>Test Name:</b>	Translate IO
<b>Purpose of Test:</b>	To verify functionality of IO component of MCMTranslate
<b>Module Interactions:</b>	Translate::IO/Translator
<b>Input Specification:</b>	Load mapping
<b>Output Specification:</b>	Build data structure of mapping description for use in translation
<b>Test Environment Restrictions:</b>	Must be tested concurrently with translator

The Translator cannot perform any sort of conversion of motion data to sound data without the presence of the mapping, which serves as the “road map” for the translation process.

<b>Test Name:</b>	Translate MIDI
<b>Purpose of Test:</b>	To verify proper operation of the MIDI output routines
<b>Module Interactions:</b>	Translate::MIDI/Translator
<b>Input Specification:</b>	MIDI commands
<b>Output Specification:</b>	Sounds or modification of current sound from output device
<b>Test Environment Restrictions:</b>	This test is performed in conjunction with a MIDI output device connected to the RT workstation.

The MIDI output routines are provided by a third-party organization, which professes the completeness and usability of their code.

<b>Test Name:</b>	Translator
<b>Purpose of Test:</b>	To verify satisfactory operation of the heuristics code in the translation portion of the system
<b>Module Interactions:</b>	Translate::IO/StreamReader
<b>Input Specification:</b>	Decoded motion data from StreamReader
<b>Output Specification:</b>	Corresponding MIDI command
<b>Test Environment Restrictions:</b>	none

This test features an individual motion data element’s information causing the generation of the mapping-defined equivalent MIDI command.

<b>Test Name:</b>	StreamReader
<b>Purpose of Test:</b>	To verify network operation of the Vicon supplied motion data reader code
<b>Module Interactions:</b>	Translate::StreamReader
<b>Input Specification:</b>	RT Emulator or Vicon 8 Camera system motion output
<b>Output Specification:</b>	Readout of coordinates for each point
<b>Test Environment Restrictions:</b>	A capture (either simulated or real) must be being performed with the data moving on a closed local area network; either the Vicon 8 camera system or its emulator can be used.

## 7. Tracking and Control Mechanisms

Since the system development for MCM is rather small in comparison to other software projects, the team as opted not to use conventional tracking and control mechanisms such as CVS or other code/update repositories.

The benefit of the separation of program elements into two parts, mapping and translation, and their subsequent breakdown into individual modules, allows each team member to cover a specific area of detail without saturating his/her workload.

The team website and account at [www.gts2k.com/~ics125/](http://www.gts2k.com/~ics125/) provides a common area for storage and transmission of all documents, code, and testing information.

The following table provides a listing of how the MCM requirements, as specified in the Requirements Specification, are satisfied by the design presented in this document. For further information, refer to the Requirements Specification Document.

<b>IMPLEMENTATION MAP</b>	
<b>REQUIREMENT</b>	<b>DESIGN MODULE</b>
3.2 Environmental Characteristics	MCM has been designed to run on the Vicon8 network in a standard Microsoft Windows NT/2000 operating system environment.
3.3.1 Correctness	All
3.3.2 Reliability	All
3.3.3 Robustness	All
3.3.4 Performance	MCMTTranslate (All)
3.3.5 User-friendliness	MCMMMap::GUI
3.3.6 Verifiability	All
3.3.7 Maintainability	All
3.3.8 Reparability	All
3.3.10 Evolvability	All
3.3.11 Reusability	All
3.3.12 Portability	All
3.3.13 Understandability	All
3.3.14 Interoperability	All
3.3.15 Productivity	MCMMMap::GUI
3.3.16 Scalability	All
3.3.18 Visibility	All
3.4.2 Transfer of Data	MCMTTranslate::StreamReader
3.4.3 Motion Data Mapping	MCMTTranslate::Translator
3.4.5 Data Range	MCMMMap::MapData

3.4.6 Pairing of Data Point Axis with MIDI Command	MCMMMap:MapData
3.4.7 MIDI Command Selection	MCMTTranslate::MIDIControl
3.4.8 Axis Mapping	MCMMMap::MapData
3.4.9 Multiple Mapping Configurations	MCMMMap::IO
3.4.10 Note Velocity	MCMMMap::MapData, MCMTTranslate::MIDIControl
3.5 UI Model	MCMMMap (All)
3.8 File Format	MCMMMap::IO, MCMTTranslate::Streamreader, MCMTTranslate::MappingInfo MCMTTranslate::Translator

## 8. Modifications to Requirements

See Requirements Document Version 1.1. Changes to the document occurred primarily because of the ambiguity in the definition “linear mappings” as related to the translation of motion data to MIDI.

Date	Version	Status/ Action	Revision By	Comments
4/26/01	1.0	Completed Requirements Specification	All	The Requirements were submitted and presented today.
5/17/01	1.1	Added more detail and clarification to the document.	All	?? 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.

## **9. Meeting Minutes**

### **9.1 - Sunday, 6 May 2001**

Evening meeting of Sal, Jimar, and Gene to finish research, work on technical presentation slides, and begin construction of the design document.

### **9.2 - Monday, 7 May 2001**

Short meeting with Professor Ebert at 12:30pm to discuss preparation for technical presentation and answer questions about topics to focus on. Sal, Jimar, and Gene present.

### **9.3 - Tuesday, 8 May 2001**

Morning meeting before class to complete technical presentation slides. Sal, Jimar, and Gene present.

### **9.4 - Wednesday, 9 May 2001**

Regular weekly meeting in the evening to go over required details for design document, proceed with individual module designs and separation of tasks. Sal, Jimar, and Gene present.

### **9.5 - Saturday, 12 May 2001**

Planned afternoon meeting for Jimar and Gene to test Vicon StreamReader code with RT emulator missed because scheduling conflict.

### **9.6 - Tuesday, 15 May 2001**

Morning meeting to coordinate slides for Design presentation. Sal, Jimar, and Gene present design concepts for MCM and related technologies in class.

### **9.7 - Saturday, 19 May 2001**

“Online” meetings to coordinate final additions and changes to design document.

## 10. Glossary (Local to Design Phase)

### API

Application Programming Interface – a set of functions that allows the developer to use the functionality of an existing application. To use the application, the developer makes the appropriate function calls. The API usually includes documentation that defines what the functions are, what the parameters are, and what the return values are.

### C3D File Format

The C3D format stores 3D coordinate and numeric data for any movement measurement, usually used in recording Biomechanics experiments.

### GUI

Graphical User Interface

### MIDI

Musical Instrument Digital Interface. MIDI is a standard protocol that was agreed upon by major manufacturers of Electronic Musical Instruments. It allows Keyboards, Synthesizers, Computers, Tape Decks and even Mixers & Stage Light Controllers to talk to each other.

### RT Emulator

A software program that reconstructs in real-time a Vicon 8 camera system motion capture event.

### RT Workstation

The Vicon company's high-powered PC running Windows NT/2000 with Vicon software that accepts the real-time capture stream on the network and outputs the current capture to a video display in real time.

### TCP/IP

Networking communication protocols for transferring data from one computer to another. The routable protocol is the standard for the Internet and it ensures reliable data transfer and congestion control.