The CAMP Music Configuration Database

Approaching The Vanilla Synthesizer

John Free and Paul Vutas
Helicon Systems Inc.
342 Byng Avenue
Willowdale, CANADA
(416)-733-1765

(c) 1988 Helicon Systems Inc.

0. ABSTRACT

The Computer Assisted Music Project (CAMP) [FREESE] is a general purpose composition and performance software environment originally based upon William Burton's SSSP. CAMP is designed to support a music lab with multiple users on a networked computer system, operating with a diverse collection of MIDI controlled, and DSP types of music equipment.

The CAMP environment is targeted at music education in schools, and therefore caters to musicians that may not be well versed in computers, MIDI, DSP technology, or the technical details of specific synthesizers. At the same time, CAMP provides flexible access to the music resources required for live performance of sophisticated pieces.

The Music Configuration Database is a fundamental component that enables CAMP to provide device independence and to make intelligent use of the available music resources. This paper describes some of the specific requirements for the database, how CAMP applications make use of it, and a brief...
description of its structure. The paper concludes with some thoughts on handling MIDI timbres in a device independent manner.

1.0 INTRODUCTION

The need for device independence, particularly within the context of the music classroom, arises from the goals of score portability, application software portability, and simplifying equipment administration.

Ideally, it should be possible to record or compose a piece using one set of equipment, and perform it on another which uses a potentially different synthesis technology. The software environment should be "smart enough" to perform the piece using the music resources available. This is especially important in a music classroom where not every student may save the same equipment. Should the resources be inadequate (insufficient number of voices, or incompatible timbre data) the system should inform the user before performing the piece.

While it may be understandable to have "patch editor" programs that are dedicated to a specific model (or class) of synthesizer, ideally application programs should be configurable to support many synthesizers. Invariably new instruments with unique features appear every year, although the MIDI access protocols seem to be standardizing within any manufacturer's line. Unfortunately software developers find it more profitable to sell a separate application for each type of synthesizer, rather than one application that handles many.

As the number of synthesizers increase, MIDI cabling can very quickly turn into a mass of spaghetti. In such a situation, it can become very frustrating to keep track of which synthesizer is connected where, and on what channel. Once the system is configured, the system should keep track of
degree of device independence that we seek. It is necessary for the software to know which of the "standard" MIDI functions are supported by a specific MIDI device, as well as a way of describing access to the "system exclusive" functions.

3.0 THE MUSIC CONFIGURATION DATABASE

The one necessary component in accommodating these needs in CAMP is a way of storing information about the various music equipment on the system such that it can be accessed by all music applications, real-time performance processes, and be easily queried and updated by users. This function is performed by the Music Configuration Database within CAMP.

The Music Administrator task [FREE1986] in the CAMP system controls the music hardware attached to that node. It is responsible for the real-time functions of recording and performing music scores, and can be synchronized with the Music Administrator tasks on other nodes. The central music database is accessed during the startup phase of the Music Administrator to read out:

- which music driver(s) should be mounted,
- what equipment is connected to that particular computer, and their MIDI channels,
- the MIDI initialization sequences for specific instruments,
- the handshaking protocol for sending/receiving timbre and instrument programming data, and
- device dependent performance characteristics of each instrument.
degree of device independence that we seek. It is necessary for the software to know which of the "standard" MIDI functions are supported by a specific MIDI device, as well as a way of describing access to the "system exclusive" functions.

3.0 THE MUSIC CONFIGURATION DATABASE

The one necessary component in accommodating these needs in CMRF is a way of storing information about the various music equipment on the system such that it can be accessed by all music applications, real-time performance processes, and be easily queried and updated by users. This function is performed by the Music Configuration Database within CMRF.

The Music Administrator task [FREE1986] in the CMRF system controls the music hardware attached to that node. It is responsible for the real-time functions of recording and performing music scores, and can be synchronized with the Music Administrator tasks on other nodes. The central music database is accessed during the startup phase of the Music Administrator to read out:

- which music driver(s) should be mounted,
- what equipment is connected to that particular computer, and their MIDI channels,
- the MIDI initialization sequences for specific instruments,
- the handshaking protocol for sending/receiving timbre and instrument programming data, and
- device dependent performance characteristics of each instrument.
CAMF has a timbre librarian that manages timbre files for specific instruments, and uses the Music Administrator task to audition and load/query voice data to/from specific synthesizers. The timbre files contain information about the destination instrument (MIDI manufacturer's ID code, a unique ID for the model number, the original program number for the voice) as well as the voice data bytes (excluding system exclusive header and handshake data). The timbre librarian uses the database to find where specific instruments are connected on the network, the size and format of voice data, and the location/format of the voice name within the voice data (if that synthesizer supports it).

The scores are represented in a device-independent manner [FREE1987], and so the score editing/orchestrating/conducting applications, as well as the various skill-tutoring packages operating within CAMF have no need to access the database. They depend on the Music Administrator task to allocate the music hardware resources, or indicate that appropriate resources do not exist. A hypothetical CAMF setup is shown below in Figure 1.
Figure 1. The Music Configuration Database Within the CAMP Environment.
2.1 User Access to Database

A graphics based, menu-driven utility called music.config affords the user the ability to browse and query the music database, and to perform common equipment management functions such as:

- adding a new synthesizer to the system,
- moving a synthesizer from one node to another,
- changing MIDI channel numbers, or
- installing a new system from scratch.

Even though the database is contained in a single ASCII text file, users are not allowed to edit the database directly. Due to the convoluted inter-relationships between items in the database, one editing mistake could make the entire music system inoperable. Music.config checks the validity of user input (e.g., MIDI channels in the range of 0 to 15, driver type is valid and the file exists) and warns the user of improperly connected devices (e.g., device connected to a node that has no interface) before updating the database. The access routine also stamps the database file with a magic number and a checksum, which makes it fairly easy for an application to detect if the database has been installed incorrectly, or has become corrupted.

Rather than have the user tediously specify the MIDI device properties for each synthesizer, music.config loads this item up from a template file. A library of template files were created for most synthesizers under common configurations (e.g., Yamaha DX7, Casio CZ101 polyphonic, Casio CZ101 4 voice) as well as a "default template" containing no
manufacturer specific MIDI features (no sustain pedal, no pitch wheel, etc). Creating a template file is done very infrequently, and requires fairly thorough technical knowledge of the specific synthesizer.

3.1 Types of Information

The database is divided into three lists of information: Interface Items (basic hardware interface description), Device Items (info on specific devices), and Properties Items (characteristics of a specific device type). Even though the lists are separate, they are interdependent.

An Interface Item informs the system whether there is any music hardware available on a particular network node, and which type of driver must be mounted to access it. CAMP has a portable driver layer for both MIDI interfaces (currently supporting the Roland MPU-401 and a "UART + timer" interface) as well as a direct synthesis DAC interface.

The list of Device Items describes each individual music input/output device on the network. It contains the device's MIDI configuration (if it is a MIDI device), and refers to a Properties Item describing the properties of the device type.

For simplicity, each device item is allowed a single MIDI receive channel, a single MIDI send channel, as well as a distinct basic channel. Thus a multi-channel MIDI device such as the Yamaha TX816 (8 MIDI receive channels) is represented as 8 separate device items in the database.

In the current implementation, only a single interface per node is allowed, and so a specific device is assumed to be attached to an interface if both items have the same node number.

The properties items describe the device dependent charac-
teristics of a specific type and configuration of MIDI syn-
thesizer (e.g. a Yamaha DX7, Roland D-50, CE-51). This
information includes:

- a (Company) id for uniquely identifying the synth type and
  model number,
- the maximum number of simultaneously sounding voices,
- number of valid MIDI program numbers and how they
  are structured into voice banks,
- the program number of the first writable timbre and
  number of writable timbres,
- whether the device supports a sustain pedal, mod wheel,
  velocity sensing, polypressure, pitchbend or aftertouch
  MIDI messages,
- which MIDI controllers are supported,
- a MIDI sequence for initializing the device,
- the size and format of timbre data, and
- the protocol for sending and receiving MIDI timbre data.

There are several reasons for keeping the properties infor-
mation separate from the Device item. It is quite common for
a system to have several devices of the same type (e.g.
these DX7's) and so some memory space is saved by storing
one property item for multiple devices. Whereas the Device
information changes every time a MIDI device is moved on the
network, the Properties information is essentially fixed for
each device type and is rarely modified.
MIDI Byte Sequences

Certain operations on devices (such as initiati ng a voice dump, or initializing the synthesizer to a specific starting configuration) require sending a sequence of MIDI bytes that is specific to that type of synthesizer.

Some MIDI byte sequences require that the MIDI channel number or the MIDI program number be embedded in the sequence, rather than "hardcoded" these values as the MIDI sequence structure uses "wildcards" embedded in the MIDI byte sequence. Thus whenever this sequence is sent, the Music Administrator task substitutes the appropriate MIDI channel and program number values obtained from the Device item for the destination device. Each byte of the MIDI sequence is in fact represented as an unsigned value, with the upper byte value representing the "Wildcard" and the lower byte the base value.

MIDI Protocol Instructions

Each type of MIDI device that is capable of sending and receiving timbre data typically has its own distinct protocol. We did not succeed in finding a flexible enough framework to describe a reasonable number of synthesizers, and so we opted for using a simple "language" with a set of # instructions (MIDI Protocol Instructions) to describe the protocol. When a timbre transfer operation is to be performed, the Music Administrator "interprets" the appropriate sequence of Protocol Instructions for that device.

The protocol instruction set is a fairly simple one: it will send and receive "literal" MIDI data or the data supplied in a buffer from the application, it has a pause instruction, and a single level loop instruction. The instruction set does not have any provision for conditional branches or loops, or expression evaluation. This makes it impossible to write an entry that causes a specific action when a specific
MIDI message is recognized, or to calculate a checksum for bulk dump message. This is a planned extension, although this feature is currently available using higher level software.

Each MIDI Protocol instruction consists of an "opcode", a numeric argument field, and a pointer to a MIDI byte sequence. The device property structure has two lists of such instructions: one for sending, and the other for receiving a timeline.

3.2 DATABASE ACCESS ROUTINES

To simplify the writing of applications and reduce the chance of corrupting the database, a library of database access subroutines were developed. All CAMP applications access the database exclusively through the use of this link-time library. This library supports the operations for:

- opening and closing the database,
- creating and deleting items within the database,
- querying items on a specific field and key, and
- loading and storing device property items from/to a template file.

Open/Close Operations

Since CAMP supports multiple users on the network, the OPEN/CLOSE mechanism uses a semiparallel control scheme to allow only a single user at a time to modify the database. Any number of tasks may OPEN the database for read-only, however once the database is OPENed for modification, all future "OPEN for modify" operations are denied until the

137  XCMC Proceedings 1988
database is CLOSED.

The OPEN operation actually reads a copy of the entire database (or optionally only those items corresponding to a specific node id) into application memory, allocating memory for each item. The CLOSE operation writes out the database (only if it was opened for modification) and then frees the memory occupied by the items.

Create/Delete Operations

The create operation allocates memory for a new device/interface structure, links it into the database list in memory, and then returns a pointer to this structure. The application can then stuff appropriate values into specific structure members.

Device and interface items remain in the database until they are explicitly deleted. If a device is temporarily disconnected from the system, it is only necessary to zero-out the node number the device item. When the device is re-attached to the system, then the appropriate members can be updated.

Query Operations

The query routines are used to sequentially search the database on a given field from a given starting position in the list. If a match is found, then a pointer to the device or interface structure is returned. A value of NULL is returned if the search fails.

For example, the query subroutines can be used to find: what equipment is attached to node 73 on the network, which nodes contain DX7's, or which MIDI channel(s) a specific device is using.

Load/Store Device Properties

Although applications may edit the device properties
directly, this situation is not very common. They typically deal with a fixed set of device types (e.g. DX7, CS10, ym-77) that are hooked up to different nodes and use different MIDI channels. To make life a little easier when installing new devices in the database, there are two routines that load and save device properties structures from/to a "template" file.

4.0 SUMMARY AND FUTURE WORK

This paper has described the motivations that gave rise to CAMP's Music Configuration Database, how it fits into the environment, and has provided an outline of its structure and contents.

The existence of a central Music Configuration Database makes possible the measure of device independence and modularity of applications within the CAMP environment and eases the task of managing a studio with diverse synthesis technologies. This database also provides a foundation from which music applications can more intelligently use the available resources, simplifying the electronic music composer's tedious tugging of MIDI channels and repatching of synthesizers. For the music classroom, CAMP provides an interface to MIDI devices that is coherent and extensible, thereby smoothing the learning process for this new technology.

Currently, the database provides transparency in data transfer between the computer and synthesizer as well as allowing device specific controllers to be mapped between synthesizers from different manufacturers. We intend to extend the database to provide higher level concepts of timbral control such as treble, bass and midrange manipulation as well as context-sensitivity. As this work progresses, we are coming closer to building the "vanilla synthesizer". Such a software module allows the composer to use one program for creating and editing each class of timbres,
regardless of how they are synthesized.
5.0 REFERENCES

