From Score to Unit Generator:  
A Hierarchical View of MIDAS

T.M. Anderson, A. Hunt, E. Kirk,  
P. McGilly, R. Orton, S. Watkinson  
Music Technology Group,  
University of York,  
YORK YO1 5DD, UK  
EEmail: nuala, adh2, ppr1, pm5, r01, nmw5@ac.york.uk

Abstract

The outline structure of the Musical Instrument Digital Audio Signal Processor (MIDAS) multiprocessor architecture for music synthesis was described at ICMC Glasgow in 1990. In this paper we describe the way in which electroacoustic music is scored and performed on the system by the musician. In particular, taking a relatively simple example, we discuss how score information is mapped through the MIDAS interpretation of the orchestra down into the algorithmic primitives and unit generators which are supported by the model of concurrency adopted by the MIDAS system.

introduction

The intention in MIDAS is to provide an extensible configuration for a range of platforms. Those will include a single host (a UNIX workstation), a host with a single accelerator co-processor (eg a DSP), or a host attached to an extended multiprocessor network, where most of the processing is carried out on the network. Emphasis is placed upon the production of software (including graphics libraries) structured in a fashion which facilitates porting between platforms, so that the total system will tolerate changes of hardware configuration.

The system was first described at ICMC Glasgow in 1990 [3]. It is a multiprocessor system designed specifically for the signal processing demands prevalent in computer music. Processing nodes are interconnected by a tightly coupled high bandwidth local area network known as MILAN. Applications are written in terms of the data communication protocol supported by MILAN, and not in terms of the instruction set of a specific processor. This means that as long as processors each have an interface capable of interpreting the protocol, several different kinds of processor may be accommodated on the network concurrently, and new processors may be added over time, giving musical systems a degree of longevity. Applications are described in terms of networks of algorithmic processes known as Unit Generator Processes (UGPs) which are distributed over MIDAS and which communicate by means of 'data packets' sent over MILAN.

The intention of this paper is to describe how a musical application would use the MIDAS system to acquire and control processing resources in order to deal with electroacoustic scores. The E-Scape system [1, 2] is used as an example application.

Scores and Instruments: E-Scape and MIDAS

E-Scape may be regarded as a high-level application for MIDAS. E-Scape is an object-oriented composition and performance environment which can use the MIDAS system as a controllable synthesis resource. E-Scape allows the composer to:

(a) graphically design instruments which contain a specification of a network of primitive processes (in this case MIDAS UGPs) which are connected to each other and to named instrument inputs.

(b) draw and edit scores. Each score event provides time-varying data for an instrument input, which must be sent to the inputs of the appropriate UGPs running in MIDAS.

235
Performing an E-Scape score therefore involves sending messages to MIDAS, to:
- specify instrument "templates" which describe a network of UGP's to be created and connected;
- request the instantiation of the number of "instrument instances" (based on the instrument template) for the required number of simultaneous events. These instrument instances will contain many UGP's which may be located in different places in MIDAS;
- start and stop an instrument instance running;
- send data from each score event to the instrument instance allocated to it. This may be in real time (as a score is played, or in live performance).

If there is to be a high density of time-varying score data, it can be downloaded to MIDAS in advance of performance. E-Scape can then command each downloaded score event to start. Alternatively, E-Scape can directly issue commands to a "scheduling UGP" in MIDAS which will then start all score events with a single command. Two sets of score events can be started and addressed as a hierarchical group.

The task of UGP's

A UGP is an algorithm comprising code which performs specific primitive operations. By 'primitive' we mean that an operation has been considered to be so fundamental as an application that a MIDAS programmer has coded it as a function. UGP's are connected in a network and data is sent between them in the form of packets.

There are two levels at which programmers are likely to want to interact with UGP's. Each level has its own view of what a UGP is and how it can be used. A UGP programmer needs to know how MIDAS operates at the lowest level so that he or she can write the code which handles the algorithmic function and the interaction between UGP's using packets. An application programmer is one level removed from this and communicates with MIDAS and UGP's via standard function calls.

UGP's are comprised of four sets of code - for instantiation, execution, termination and configuration. Many copies of a particular UGP may be required for a single score. The job of the instantiation code is to create data areas to be used by each copy. The execution code uses the data areas specified for each instantiation. Termination code deals with tidying up when an instantiation has run its course and is no longer required. Configuration code allows networks of UGP's to be interconnected and permits run-time changes to UGP's - for example the number of inputs a UGP may need can be altered while it is running. Application programmers need to be able to create a number of UGP instantiations, connect UGP's together, pass data to them, and finally delete no longer wanted UGP's.

From the viewpoint of a UGP writer MIDAS provides a framework deliberately comparable to Csound [Varela]. Therefore the translation of UGP's from Csound is relatively direct. The main difference is that since MIDAS is not a shared memory system, the result of a UGP must be processed by a library function that directs the data to the appropriate node. All details of networking, message addressing and message routing are hidden, giving the user the impression of a single processor system.

From the viewpoint of an application programmer MIDAS is a C-Library. The library functions handle all of the communications with the network. Functions of the following form are provided:

```c
create_UGP(UGP_id, UGP_type)
connect_UGP(ptr_UGP_id, dest_UGP_id, input_no)
sendval(UGP_id, value)
```

Interfacing Scores and Instruments with UGP's - MII

E-Scape communicates with MIDAS in terms of instrument specific events and score events. Within MIDAS itself, however, there is a network of intercommunicating UGP's distributed around the system at various processing nodes. The two systems cannot directly communicate with each other as they are intended to be separate entities, capable of working with other applications. This is an important design consideration, as authors of high-level applications should not have to be concerned with the low-level implementation of UGP's. On the other hand, MIDAS is designed to have a small and portable command set, and thus should not be expected to cope with the complex demands of a high-level application designed for composers. Thus there exist the need for a translator process which can interface between the user application and MIDAS.

The MIDAS Intermediate Interface (MII) receives descriptions from the E-Scape application about the UGP's
required in each instrument template, and how they are to be connected. When MIC receives a 'Create Instrument Instance' message, it requests the instantiation and connection of each UGP within the instrument template. This is repeated for each instance required and MIC keeps track of all updated MIDAS addresses in a dynamic database.

MIC can also receive score information downloaded in advance of performance, which is accommodated in scheduling UGPs. Time-stamped information packets are stored within the scheduler UGP, having been pre-addressed to a specific UGP input in MIDAS. When the scheduler UGP is triggered, it sends out these packets to the appropriate locations at the correct time. MIC is also able to instantiate and control UGPs as a response to real time performance requirements.

Current Implementation of MIDAS

MIDAS has been implemented on a network of SGI Indigo workstations using the UNIX 'sockets' facility. Instrument porting to a transputer array is expected, followed by a hardware implementation on a custom network of DSP96002 processors.

The system is modularized such that the machine dependent code is contained in one small library, allowing simple portability. The libraries provided ensure that the multi-processor aspects of the system are completely transparent, enabling the user to concentrate on the more important activities of instrument and score design.

The current implementation is in three parts: the Command Line Interface (CLI), Server, and UGPs. The UGPs are written in n-entrant C functions that use no persistent local variables. A pointer to the complete environment for the UGP is passed as a parameter on each call. A server runs on each node accepting messages from, and delivering messages to, the network. It also manages all UGPs on the node, running them in accordance with the dataflow methodology. A dataflow system is a process only invoked when there is data available on all of its inputs. Thus the availability of data provides synchronization. The operation of UGPs is independent of their location on the network and connectivity. The CLI, running on the host, allows the user to create, connect, configure and delete UGPs, and to send messages to them. Similar operations can be performed on tables.

Instruments can be constructed interactively from the query console with the CLI, transmitted from an external application, or loaded from a script file. Timed data messages may be used to transmit information to UGPs. Audio data can be transferred to and from disk or audio ports on suitable machines such as the SGI Indigo.

The capabilities of MIDAS in supporting distributed, frequently communicating processes in finding applications in other fields: for example, general signal processing, video processing and road traffic flow modelling.

Conclusion

Our purpose in carrying out the work described in this paper has been to formulate an exercise which would require us to assess critically the design principles which form the basis of the MIDAS system.

We are encouraged that our original concepts have found a stable and viable basis for the support of electroacoustic music within the E-Scape and MIDAS environments. The MILAN protocols have also proved to be sufficient for this purpose, and so the goal of portability through their use also seems viable.

When the implementation of the protocols and UGPs stabilizes, we hope to be able to release this material into the public domain, following the enlightened example of Vorrс's CSound. Development may then continue within a number of institutions.

References


237
Hierarchical View of ESCAPE application configured on the MIDAS system via MII