Evaluation of Architectures for Sound Generation Systems with Respect to Interactive Gestural Control and Realtime Performance

Paul Modler  
Music Department, University of York  
Heslington, York, YO10 5DD  
plpm1@york.ac.uk

Ross Kirk  
Department of Electronics, University of York  
Heslington, York, YO10 5DD  
ross@ohm.york.ac.uk

0 Abstract

With the emergence of new and sophisticated control devices like data gloves and data suits, there is an increasing need to integrate gestural expression into the musical composition and performance environment. In such an environment emotions can be expressed through sensor devices and associated musical software. State of the art sound and music generation systems like MAX/FTS, MIDAS, PD/GEM and Supercollider can be used for these applications. The issues and related demands such as visualisation and scripting addressed by these software packages are complex and resource intensive. The underlying software architectures therefore have to deal with multitasking, parallel processing, real-time operation etc. The aim of this Paper is to evaluate architectures of known software and hardware systems with respect to their ability to integrate gestural control algorithms and devices into a sound and music generation environment.

1 Sound Generation Systems

1.1 FTS/Max

FTS/Max is widely used for real-time sound generation, transformation and analysis. The initial implementation was on a NEXT machine with additional processor cards (ISPW). Implementations are also available on SGI platforms, including the Origin 2000. FTS/Max is constructed in 2 layers: FTS as the sound processing real-time kernel and Max the superimposed graphical user interface.

FTS provides the necessary facilities for the processing of sound data in real-time. It provides a user programming interface which has to be used to develop new sound processing objects. The user programming interface is based on C functions, which provide an object oriented programming environment. This brings the drawback of non standard C++ object environment, but the advantage of pure C portability, which could be used to port code onto signal processing hardware platforms, which provide mostly a C Compiler. The following description is based on the SGI Origin implementation of FTS.

The current release of FTS does not support parallel processing on an Origin2000 in a way comparable with the earlier ISPW version. Although it is possible to run several FTS applications in parallel as daemons, and have them communicate through the message passing tools provided, there is no ready to use situation for the use of the multiple processors of an Origin.

1.2 MIDAS

The MIDAS architecture is constructed as a real-time multitasking kernel, with a built in protocol for parallel processing applications (Kirk, Hunt 1996). Facilities for applications running on multiple workstations or processors (eg DSPs) are provided for message passing, audio data streaming and synchronisation. User programs are constructed in form of uggs which are linked to the MIDAS kernel. Sound and gestural processing algorithms are constructed dynamically at run-time using protocol elements designed to instantiate (and delete) uggs on specified processing nodes. MIDAS is written in C and is thus portable to different platforms including signal processors. A first stage graphical user interface for creating networks of uggs in a Max style manner exists.

The system is able to include graphical applications, although no special packages exist. OpenGL is widely used to add graphic uggs to the system.

1.3 PD/GEM

PD is a recent sound-processing and analysis tool by Miller Puckett who also initiated FTS/Max. The architecture is based on FTS. The authors attempt to provide compatibility with applications generated on the FTS/Max system. GEM is a special package for integrating graphical applications into the PD environment. PD and GEM are written in C and therefore are portable to different machine platforms. PD/GEM is not at present intended to run on parallel processing machines.

1.4 SuperCollider

SC is a sound-processing tool which runs on PPC machines. It is constructed as a real-time kernel providing processing of sound and graphics. Source code is written in the Supercollider language and is compiled to run on the real-time kernel. The language provides object oriented programming, but function-oriented programming is supported as well. A large set of list oriented processing commands offer LISP style programming. In short the language can be described as a crossover between Smalltalk, LISP and Java. The following layers of the System can be described:

- audio rate level (audioprocessing)
- control rate level

Although the user can use the comprehensive set of audioprocessing objects, SC does not support a low-level C programming interface for adding user defined audio processing objects.

ICMC Proceedings 1999
A broad set of graphical primitives is provided to combine graphical and sound processing. A graphical Max style user interface is not provided.

1.5 UNIX

Although Unix is not a realtime processing system itself, it provides utilities for processing gestural data. Scripts can be used to run different applications connecting them through pipes or sockets. Since there is no overhead from an environment like Max/FTS applications can run faster directly on Unix. The user has direct access to system resources. With load balancing tools like Share II processes can be scheduled to run on a specified number of processors.

Unix provides several mechanisms for communication between program units (processes and threads). Sockets and pipes can be used to pass information between processes. Information can be passed through sockets onto other processors or to other machines. To exploit the parallel features of the system as far as possible users are responsible for writing their code in an appropriate way.

This can be achieved by constructing the software in threads, which are process like chunks of code but using the same address space and process state. These threads can be distributed over the multiple processors of a system. By these means users are free to draw all benefits of the parallel architecture of the Origin, but also have the burden to program this themselves.

With the concepts of Shared Memory (OpenMP) or message passing (MPI) code can be parallelised down to the loop level. Users have an additional burden to put the necessary directives into their code but will improve system performance by this means.

2 Architectural Demands for Gesture Processing

It is useful to consider the requirements of gestural processing which reveal appropriate pre-requisites for the underlying processing architectures. We identify the following characteristics of gestural processes which might form the basis of evaluation criteria for these architectures: synchronisation with synthesis algorithms; processing and communication network latency; integration of gestural and synthesis algorithms (multiparametric mapping); distributed processing and scalable run-time performance.

It is important to realise that the complexity and performance requirements of gestural processing algorithms may be comparable with those of the signal processing algorithms forming the core of the sound synthesis system. Gesture processing algorithms based on Neural Nets and Hidden Markov Models will probably use a multiply-accumulate operation similar to that used in digital filters used in synthesis for example. Furthermore, the recognition algorithm may need to be closely and directly integrated into the synthesis algorithm. Investigations reported in [Hunt, Kirk 1999] indicate that the direct one-to-one mapping of gestural to synthesis parameters commonly found in interactive systems may not be efficient in sound synthesis applications. Highly integrated one-to-

3 Analysis of Musical Systems against Architectural Demands for Gestural Processing

The control of a synthesis system by gestures is accomplished by using one or several interfaces, digitising the body movement into data. A wide range of interfaces have been built and many are commonly available as devices which use the MIDI protocol as the digitisation format. This enables the device to be connected to most of the commonly available sound synthesis systems. The systems described in section 1 offer state of the art MIDI implementation and therefore are suited for such devices. In all of these systems the stream of incoming MIDI data can be mapped to sound synthesis parameters. Whereas in Max/FTS this is done through the graphical programming interface, in MIDAS the user has to code the mapping of the MIDI stream to the control parameters. Several issues compromise the use of MIDI as a control media for sound synthesis. These are related to:

- The time Granularity of the MIDI stream
- The granularity of the sound synthesis parameters, caused by MIDI protocols
- The design of MIDI drivers for multi-parametric input devices like data gloves or sensor suits.

It is therefore appropriate to consider the systems described in section 1 against the architectural criteria identified in section 2.

3.1 Max/FTS/PPD

Synchronisation with Synthesis Algorithms: Max/FTS provides user definable clockrates, thus enabling it to synchronise with the gestural stream, which is not necessarily presented at audio rate level or control rate level. It is important to realise that the integration of recognition algorithms require precise timing of the
gestural stream, both during the learning and the recall phase.

Processing and Communication Network Latency: Max/FTS supports network communication for Max messages. Max can be hooked to an FTS daemon running on a separate host, providing transmission of messages from Max to the FTS server and vice versa. Audio streaming is not supported in such a manner. Using the application programming interface a user defined protocol can be established using sockets to pass gestural data through a TCP or UDP socket. In terms of latency a UDP socket is preferable. With such an extension, gestural data can be transmitted over a network. The latency is thus determined by the performance provided by a UDP connection.

Integration of gestural and synthesis algorithms: Through its application programming interface gestural algorithms can be implemented in standard C and linked to the system as a DLL library. The gestural algorithm would be implemented as a standard FTS object. For the data stream an appropriate protocol needs to be defined, since the audio or control rate streams provided by Max/FTS are not necessarily adequate.

Distributed Processing and Scalable Run-Time Performance: Although FTS is multithreaded, parallel processing and scalable runtime performance is not provided on an Origin 2000 at the moment. With the application programming interface a gestural stream could be established between different FTS processes running on an Origin 2000. An internal use of the parallel features of the Origin 2000 would be more desirable.

3.2 SuperCollider

Synchronisation with synthesis algorithms: Synchronisation in SC can be achieved through user definable semaphores. As in Max/FTS the user has to define a protocol for the gestural data.

Processing and Communication Network Latency: Since SC is up to now a standalone system it doesn’t provide network communication.

Integration of Gestural and Synthesis Algorithms: Gestural algorithms like neural networks can be integrated into SC using its own language. SuperCollider’s comprehensive set of list manipulation functions can be used to program vector processing. Due to the lack of a low level programming interface previously coded software has to be rewritten in SC.

Distributed Processing and Scalable Run-Time Performance: As mentioned earlier SuperCollider is up to now a standalone application. With the announced Windows version of it and a multiprocessor machine a scalable architecture could be achieved in the future.

3.3 MIDAS

Synchronisation and Integration with synthesis algorithms: In a MIDAS implementation, a gestural processing algorithm would be encoded as network of units, in exactly the same way as a synthesis algorithm. This means that it is unnecessary to draw any distinction between the two functions from the point of view of synchronisation and integration. The MIDAS architecture is able to provide synchronisation down to the granularity of the audio sample where necessary, and the protocol structure is capable of building fully integrated algorithmic networks for synthesis or gestural purposes, or both.

Processing and communication network latency: MIDAS runs as a systolic system. The load distribution policy allocates upgs to processing nodes in such a way that the whole network can be run within one systolic cycle. In a typical system, this would be set to the audio processing rate, although multirate processing (e.g. one rate for gesture processing, another for synthesis processing) is also possible.

Distributed processing and scalable run-time performance: The MIDAS protocols allow one logical upg network to be distributed across as many processing nodes as necessary, assuming that the interprocessor network bandwidth is capable of supporting the resulting inter-upg network traffic. More processing nodes can be added as required, to give scalable performance, although speed-up factors will be impacted by the load allocation topology.

3.4 SG Origin 2000

Synchronisation with synthesis algorithms: Using the Origin 2000 as a platform for gestural processing and sound generation requires gestural processing to be done in the same process as the sound generation algorithm. The user deals with synchronisation using semaphores or equivalent mechanisms. If the user connects different applications through pipes or sockets the influence of the UNIX architecture comes into play. The throughput and the performance of the application depends on the workload of the system, as well as of the prioritisation of the different processes.

Processing and communication network latency: On a networked UNIX application for gestural processing the communication performance depends on network constraints and the communication mechanisms used. For instance, as stated above a UDP connection can be used to provide a stream oriented transmission of gestural data.

Integration of gestural and synthesis algorithms: On the UNIX level the integration of nonstandard devices for gestural input comes down to writing an appropriate device driver. The integration with other sound generation algorithms need to be more sophisticated, since no standardised protocol or piping interface exists. Users have to implement their own protocol.

Distributed processing and scalable run-time performance: The Origin 2000 architecture supports multiple processors. The current version of the IRX operating system supports dynamic distribution of processes to the different processors. Special load-balancing toolkits can be used to customize this procedure to the needs of the application. Since the Origin architecture is extendable up to 128 processors the system can be adjusted to the demands of the application if needed.


4 Towards a Unified Architecture for Gestural Processing

All of the systems described in this paper conform to the architecture described in figure 1 to some extent, which may consequently form the basis of a unified architecture. The gestural data stream is fed into a processing section which handles feature extraction, gesture recognition, mapping, synchronisation and post processing (gesture interpretation and mapping onto consequential behaviours). If the underlying processing infrastructure is capable of delivering guaranteed timing performance, then gestural streaming can be used to integrate gesture with synthesis. If the delivery mechanisms of the infrastructure is less deterministic, then a more pragmatic approach involving the time-stamping of a frame oriented protocol may be used. This is still compatible with the streaming mechanism - it is only necessary to incorporate head and tail tags into the streaming protocol packets. Gesture information gathered from feature extraction during the learning phase is stored in local memory, forming a database of gestural semantics during the training phase. The postprocessing section analyses the output of the gestural processing section and interprets the results of the gesture recognition in the light of the content of this database. This could include a state machine for building gestural semantic phrases from the recognised gestures. The gestural stream is thus connected through the processing and the postprocessing section to a control stream, or directly to the audio stream.

5 Conclusion

All of the systems described in this paper conform in principle to the architectural model described in figure 1. We have also identified a set of processing characteristics necessary to support gestural processing: provision for synchronisation, integration, distribution, scalable performance and minimal latency. However all the systems considered are deficient in some respects. Max/FTS in its standard form lacks facilities for distributing processes on the Origin 2000 architecture with a resulting lack of scalable performance. SuperCollider currently exists as a stand-alone configuration with no network utilities. It also lacks low-level programming interface mechanisms. Current implementations of MIDAS provide only primitive configuration and object oriented programming tools. UNIX is not an integrated sound processing environment in the same sense as the other systems discussed here. It is essentially a platform from which such applications can be constructed. Whilst many of the algorithmic primitives required are provided in some form, it is difficult to guarantee hard real-time constraints in many cases. The protocol primitives may also have to be developed to accommodate gestural devices. A useful model has been developed for gestural processing, however further investigations need to be carried out to provide experimental results using the architecture in musical performance.

6 References


ICMC Proceedings 1999