PARSYNTH: A Case Study on Implementing a Real-Time Digital Audio Synthesizer

Byron Jeff
Department of Computer and Information Science
Clark Atlanta University
Atlanta, GA 30314
Email: byron@cc.gatech.edu

Karsten Schwan
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
Email: schwan@cc.gatech.edu

Abstract

Current real-time systems with heavy computation loads and unpredictable event sequences are being constructed with mechanisms and policies that permit them to adjust to new environmental conditions. We present a framework for the real-time synthesis of multimedia artifacts and an application, PARSYNTH, which utilizes the libraries specifed in the framework. Our results indicate that PARSYNTH generates digital audio artifacts within real-time constraints.

1 Motivation

Real-time systems interact with their external execution environments, and complex, large-scale systems often cannot anticipate all such interactions. As a result, real-time systems are increasingly being constructed with mechanisms and policies that permit them to adjust to new environmental conditions, based on higher-level knowledge about system functionality and requirements[3, 1].

In addition the area of real-time scheduling has received increased attention during the last few years, in part due to the creation of many recent multiprocessor and distributed applications demanding real-time performance. Such applications include robotics , multi-media, and music . These applications are characterized by their need for real time response, heavy computational load, and unpredictable events sequences that must be processed in a timely manner.

The specific application domain addressed by this research, Dynamic Interactive Multimedia ArtifactS (DIMAS), concerns multi-media systems. Specifically, DIMAS provides a framework for the real-time synthesis of voice, music, graphics, and video where partially ordered sequences of computations and communications must be processed under given real-time constraints. This framework consists of A library of multimedia generation and manipulation objects and the linkages between these objects, language support for specifying interconnections between existing library objects and the ability to create user defined application specific objects and links, and run time support to execute a DIMAS constructed application in real-time on shared memory multiprocessor computers.

DIMAS utilizes the Dynamic PArallel Real-Time Scheduler (DPARTS) to perform (on-line) scheduling of action sequences triggered by real-time events, where multiple action sequences and different independent segments of a single action may execute in parallel. The PARSYNTH real-time parallel digital audio synthesis system is an application using the DIMAS run-time libraries.

2 DIMAS SYSTEM OVERVIEW

The goals of the DIMAS system are to provide mechanisms, policies, and human interfaces for dynamic system configuration and control, and to provide a framework for the execution of multi-media, real-time applications on parallel and distributed target computing platforms.

Traditional computer based multimedia generation systems have shortcomings that limit flexibilty and creativity [2]. These fall into two broad categories:

Hardware based and software based generation systems. Hardware systems typically are limited in types of patches and number of simultaeous events while software systems typically do not deliver real-time performance.

2.1 DIMAS Components

2.1.1 DIMAS Objects

DIMAS structures interactive multimedia applications as sets of interacting objects, which implement the performance and production agents mentioned above. For example, a typical real-time sound synthesis application can be broken down into components such as oscillators, mixers, and filters, which communicate with one another via sample passing.

The components mentioned above translate naturally into program objects. The connections between these "black boxes" translate into interactions between program objects. Furthermore, the object-based approach affords a straight-forward approach to parallel programming, since program objects are a natural unit of computation to distribute among multiple processors. As a result, DIMAS applications are constructed from a set of audio and graphics generation objects.

To maintain flexibility for execution each object is designed to process a small amount of work for each work request. The basic model for a object is as follows:

```
    Initialize
        |
        |
        v
--> Wait for new work request on my admin
|   link.
|       |
|       v
|   Process packets from work request.
|       |
|       v
|   Send packets down output link(s).
|       |
|       v
|   Inform object at other end of link of
|   new work using its admin link.
|       |
|--------
```

A DIMAS object accepts work from any entity that is able to send administrative packets to the admin link for the object. This provides a simple linkage mechanism for the DPARTS scheduler to control DIMAS objects. The scheduler sends work requests to each of the objects specifying the amount of work each object is to perform. CPU utilization for each objects can be varied by specifying differents amounts of work for each iteration through the execution loop. In addition the scheduler can receive information from objects via a simularly constructed administrative link. The object execution loop model is designed to facilitate the interfacing of DIMAS objects and the DPARTS scheduler.

2.1.2 DIMAS Links

Links embody the interaction semantics between objects. Namely, program object instances are connected by link instances, and links contain detailed descriptions of user-defined invocation semantics.

From the point of view of an object which uses links to interact with other objects, the links are themselves objects which export methods that implement the interaction. Simple links however can be implmented without defining specific objects but by simply rerouting the invocation/message from one DIMAS object to another.

A primary advantage of the link concept is that it allows the programmer to avoid explicitly naming other object instances in the description of an object class which is intended for reusability. An object does not have any knowledge of the objects which produce its inputs or consume its outputs. In addition an object can dynamically be instructed to accept or drop additional links at runtime.

2.2 Constructing DIMAS applications

Constructing a DIMAS application can be thought of as proceeding in several phases:

Phase 1 - the programmer writes class descriptions for any required new program object classes, and in some cases for new link types.

Phase 2 - the programmer uses language-level tools to create program object instances, connect these instances using instances of various types of links, and assign and analyze timing attributes and parallel execution attributes.

Phase 3 the programmer adds user-interactive object instances, thereby essentially designing an application-specific user interface.

Phase 4 the programmer uses graphical or MIDI-capable user interface DIMAS objects to interact with the running DIMAS program.

DIMAS application development is simpler than the development of arbitrary parallel object-based programs because built into the DIMAS system are a collection of audio and graphics generation objects along with graphical and device interface objects. A composition is constructed by assembling a group of audio-video generator objects. These objects produce sound/images based on event inputs from input devices external to the DIMAS application which interface to the application via the DIMAS device interface objects. External users can interact with the DIMAS application via DIMAS user interface objects.

Another aspect of the DIMAS system paradigm is that once system performance are consistently maintaining real-time levels that the actual artifact generated by the system need not be maintained by the system after execution. Since the specifications of a composition (including objects, patches, events, scores, visual objects, links) are often several orders of magnitude smaller in size than the artifact produced by the execution of the system, a significant reduction in the storage and transmission requirements of a composition can be achieved.

3 A DIMAS APPLICATION: PARSYNTH

A parallel digital audio synthesizer (PARSYNTH) serves as an application using the DIMAS run-time libraries. PARSYNTH is an example of the type of application produced at the end of the DIMAS application specification compilation phase describe above. PARSYNTH has been implemented with a message passing link mechanism that represents the links in the DIMAS model. PARSYNTH also implements the DIMAS object model that simplifies the linkage of the PARSYNTH application to the DPARTS scheduler. The PARSYNTH implementation serves as a testbed for demonstrating concepts and models embodied within the DIMAS system.

PARSYNTH requires several audio representations to do its job. PARSYNTH produces CD quality audio at a sample rate of 44.1 Khz. The digital audio entities based on this representations are:

- sample - a single unit of digital audio. As stated above PARSYNTH must product 44100 samples for each second of sound generated.
- packet - The smallest unit of digital audio that PARSYNTH uses. A packet currently consist of 21 samples. PARSYNTH produces 2100 packets per second. All of PARSYNTH's timings are based on packet numbers.

3.1 PARSYNTH IMPLEMENTATION

3.1.1 COMPUTATIONAL IMPLMENTATION

PARSYNTH is implemented with the DIMAS object run-time libraries.

The C-threads parallel threads library provides a lightweight threads environment for implementing objects. The DIMAS run-time object library is implemented in C-threads. Each object is assigned to a single thread with multiple threads assigned to each available processor in the multiprocessor machine.

3.1.2 OBJECT TYPES

Currently 5 types of objects are implemented:

Generators: This type of thread generates a digital audio stream.

Syncronizer: This type of object is required when multiple audio links are directed into a single object. Since generators function asyncronously and there is some latency involved between the receipt of a work request and the actual generation of packets, often the packet numbers between different streams vary. This object's job is to receive multiple links of input, collate the links into syncronization by matching the packet numbers of the packets on each link , and pass the syncronized packets of each link into the output links.

Combiner: This type of object combines multiple links into a single link. By convention the incoming links must be syncronized by a syncronizer object. The first implemented combiner object is an digital audio mixer.

Output: This object type has the specific purpose of directing the final digital audio link produced from PARSYNTH to the outside world.

Due to the large size of the output streams generated, the implemented output object actually sends the stream to a Unix socket. An external applications can then direct the stream to the digital audio output device or to a file.

Dispatch: This type object is unique in that it doesn't directly process any of the digial audio streams in parsynth. Dispatch object monitors for new incoming events, introduces new events into the PARSYNTH system, and monitors the packet numbers output from the output object.

3.1.3 COMMUNICATIONS IMPLEMENTATION

Objects in PARSYNTH transmit data between one another using links. PARSYNTH uses shared memory, which is implemented in the C-Threads Library, to implement both packets and links. Inter-object communication is accomplished through a separate link called the admin link. Each object receives work requests via its admin link. Objects communicate with other objects by sending messages to the destination object's admin link. The types of admin messages sent are as follows:

- NOTE ON
- NOTE OFF
- NEW WORK
- CONTINUE
- INFO
- DIE

4 DIMAS Run-time Scheduler: DPARTS

In order for DIMAS applications to achieve real-time performance under varying external event input loads, a dynamic scheduling system must be utilized at run time. The DIMAS framework along with the underlaying DPARTS parallel real-time scheduler utilizes the enhanced computational power of parallel distributed computing platforms to give real-time performance for software based multimedia applications.

In addition DPARTS utilizes application specific information in order to make scheduling decisions. For example in digital audio applications perceived lateness is tied to the initial attack rate of a note. For example sounds with fast attack rates, such as percussive and piano sounds, are perceived as being late sooner than a softer attack, such as strings. The DIMAS application designer can embody such attributes in the object specification and the DIMAS objects share this scheduling information with the DPARTS scheduler. So in the instance that an event with a hard attack rate and one with a soft attack rate are presented to the DPARTS scheduler at the same time, the scheduler will schedule the event connected to the hard attack rate before the event with the soft attack. Even in the case where the soft attack event misses its dealine, the perception of lateness by the listener is lessened due to the nature of the note.

DPARTS schedules and invoke groups of DIMAS objects based on event inputs from DIMAS objects that interface to devices external to the DIMAS application. DPARTS must be dynamic because it has to modify the schedule based on the available time and resources that the DIMAS objects have to do their jobs. Because of this feature DPARTS may schedule only certain features of a DIMAS object if the time and/or resources are not available for a completion of that object's task. Part of DPARTS task is to provide information to DIMAS objects about resources available and instruct DIMAS objects to use reduced functionality because of limited resources.

In order to achieve real-time response, the DPARTS scheduler is internally concurrent so it can be easily scaled to different size parallel machines and to varying application demands. This implies that scheduling is performed by multiple concurrent and cooperating objects.

As stated above, each external event may trigger a sequence of objects that jointly process the event. These objects must execute in an application-specific order. DPARTS must have information about (1) such orderings, (2) the computation time of the involved objects, and (3) about possible alternate or optional object invocations. We will develop a directed graph representation to contain such information, where the computation times and any other information DPARTS needs to know about objects is stored in the graph's nodes, and the directed edges represent the processes' order of computation. Objects must be scheduled concurrently and to meet real-time constraints, such that the graph's topological order is maintained. This implies that multiple parts of the schedule graph may be active at any point during the applications execution.

5 Initial Results

All testing was performed on an SGI Challenge 4 CPU platform using a 4 instrument MIDI file that generates 60 seconds of digital audio. The PARSYNTH application consists of 8 objects: 4 sine wave generators, a syncronizer, a mixer, an output object, and the dispatch object.

The initial result using a single processor multi-thread execution run completed in 31.5 seconds. However the initial run of the 4 processor multi-thread execution run took 234.31 seconds to complete.

An examination of the trace file generated from Cthread's monitoring facilities showed that for the multiprocessor run that the threads processors were idle much of the time due to light computational load.

As an initial test of this hypothsis the packet size for PARSYNTH was increased by a factor of 10 (from 21 samples per packet to 210 samples per packet).

The results improved. The single processor multi-thread execution run completed in 17.5 seconds and the 4 processor multi-thread execution ran in 41.47 seconds.

These results point towards three trends: The overall performance of the system can be controlled by varying the grainularity of the packet size. The multiprocess application needs heavier workloads to be effective. Some fine tuning is required to reduce the computation time of the multiprocessor application to match and exceed the computation time for the single processor version.

6 Acknowledgements

Vernard Martin assisted in editing and with the monitoring of PARSYNTH. This work is supported in part by the Office of Naval Research under grant N00014-93-1-0783 and in part by ARO grant # DAAL03-92-G-0377.

References

[1] S. Davari, T. Leibfried, S. Natarajan, D.Pruett, L. Sha, and W. Zhao. Real-time Issues in the Design of Data Management for the Space Station Freedom. In *Workshop on Real-Time Applications, New York*, pages 161–165. IEEE, May 1993.

[2] B. Jeff and K. Schwan. Dimas-constructing dynamic interactive multimedia artifacts. In *Workshop Proceedings: THE ROLE OF REAL-TIME IN MULTIMEDIA/INTERACTIVE SYSTEMS*. IEEE, 1993.

[3] J. McDonald and K. Schwan. Ada Dynamic Load Control Mechanisms for Distributed Embedded Battle Management Systems. In *First Workshop on Real-time Applications, New York*, pages 156–160. IEEE, May 1993.