An Electronic Laboratory for Operating Systems and Computer Networks

Steven Robbins

March 6, 1998
5:13 am

Contents

1 Results of Previous NSF Support

I have been an investigator on one NSF grant in the past five years: ILI USE-0950407

Amount: $ 91,000

Duration: May 1990 – October 1992

Principal Investigator: Kay A. Robbins

Co-Principal Investigators: Steven Robbins and Neal Wagner

The objectives of this project were the creation of a state-of-the-art laboratory and the development of a model laboratory curriculum for introducing concurrency in the undergraduate curriculum. A Unix workstation-based laboratory was successfully established and maintained. Six major projects and more than a dozen smaller projects were developed as part of this curriculum. All of the projects explore topics in concurrency, communication and synchronization without requiring extensive programming. Each major project consists of four to eight progressively more difficult laboratories. The laboratories incorporate fork, exec, pipes, sockets, remote procedure calls, streams, signals, mutex locks, condition variables, semaphores, shared memory and threads. For the past six years the principal investigators have taught the undergraduate operating systems course and used this material. Additional projects have been developed for the undergraduate networks course. Advanced versions of the projects have been used in the graduate operating systems course during five different semesters. Student comments have been very favorable.

This project has had substantial local impact both in terms of curriculum and because the laboratory serves as the primary computing facility for all computer science majors starting at the sophomore level. Prior to the establishment of the laboratory facility, computer science majors used terminals and connected to a Unix server at the University of Texas at Austin, 80 miles away, through an unreliable network connection.

The investigators have presented their work at SIGCSE and ASEE conferences. They have written a reference book on this work which was published by Prentice Hall Professional Technical Reference in November 1995. This book has been used at a number of institutions including MIT, Stanford and USC. The book provides a step-by-step guide to developing concurrent programs. In addition to nine topic chapters describing the fundamentals of process and device management, synchronization, threads and communication, the book has separate chapters devoted to each of the six major projects developed as part of this grant. It is too early to accurately assess the national impact of this work which will depend, to some extent, on how widely the book is used. Early reviews have been very favorable. The following publications have resulted from this work:

1. K. A. Robbins and S. Robbins *Practical Unix Programming: A Guide to Concurrency, Communication, and Multithreading*, Prentice Hall, 1996.
2. K. A. Robbins and S. Robbins, "Blackboards for cooperative problem-solving," *Proc. ASEE DELOS Conference*, (1993) pp. 459–465.
3. R. Rybacki, K. A. Robbins and S. Robbins, "Ethercom: A study of audio processes and synchronization," *Proc. 24-th SIGCSE Technical Symp. on Computer Science Education*, (1993) pp. 218–222.

See `http://vip.cs.utsa.edu/pup` for code and additional information.

2 Problem Statement

The geometric increase in computer power and the ubiquitous availability of network connectivity are revolutionizing computing. Computing trends are toward sophisticated data sharing, realtime interaction of applications, intelligent graphical user interfaces and complex data streams that include audio and video as well as text. This new technology can enhance the teaching of computer science, but for the most part it has not been used to its full capability.

A computer science curriculum based on old teaching techniques may seem dull and irrelevant to the modern student. Instruction in the undergraduate curriculum for the most part does not make *active use* of new technology. A multimedia presentation during a lecture by an instructor is considered to be a passive use of technology. In contrast, a student who can control a multimedia visualization is making active use of such technology.

The difference between passive and active use of technology is similar to the difference between watching a laboratory demonstration and performing an experiment. Aside from making the learning experience more interesting, a well-designed experiment can challenge the student and encourage creativity. A user-controlled animation can have a similar effect and provide the student with a virtual laboratory environment.

3 Specific Objectives

The goal of the proposed project is to significantly enhance the undergraduate curriculum in operating systems and networks by using visualization, animation and auralization in an active way. Visualization refers to using images to enhance understanding, while animation refers to the use of visual displays that change over time. Auralization refers to using sound to enhance visualization or animation.

I plan to completely implement a hands-on approach to operating systems and networks. A significant start in the operating systems area has been made. A colleague and I have developed an extensive set of projects to illustrate synchronization and communication [28]. The areas addressed so far involve topics that are best learned by writing programs. Writing programs is time consuming and there is not time for more than a half dozen programming assignments in a semester. Some topics in operating systems are better illustrated in other ways. These include classical synchronization problems, process scheduling, memory management and I/O scheduling.

I will also build on my previous work in animating network protocols to design a more hands-on approach to computer networks. A detailed hands-on curriculum will be developed to illustrate several major topics in computer networks.

This project will produce a collection of interactive animation projects to enhance the undergraduate curriculum of operating systems and networks. Each animation will illustrate a concept and encourage the user to examine the concept in depth. These animations should improve the student's understanding of the concepts and make learning more enjoyable and challenging.

My research into algorithm animation has led me to develop JOTSA, an animation package written in the Java language. JOTSA provides an ideal development environment for the proposed project. JOTSA programs can be run on any machine (PC, MAC, workstation) with a browser connected to the Internet, so the products resulting from this project will be universally available.

By the time this project starts in fall 1997, JOTSA documentation will be complete enough for others to use it. I have developed a prototype of one animation project in JOTSA to prove that the concept is sound. The prototype explores a set of network protocols and is being used in the computer networks class I am currently teaching. (See Section 5.5 for additional details.) Under this proposal, this prototype will be enhanced to support active use by students. I have many ideas for incorporating such active venues and for additional animation projects related to the operating systems and networks curriculum. (See Section 5.4.) However, developing these ideas into a project that can be used in a course, let alone refining them so that they can be used by others, is a time consuming task. It would not be possible for me to do this without assistance even if my other responsibilities did not interfere. Even with the assistance of the two graduate students supported by this grant, creating the number of interactive animations required would be difficult. As insight is gained by work on additional animations, I will develop a framework for producing interactive animations that will increase our productivity.

The purpose of the proposal is give me the time to organize this project, develop a framework for interactive animations, and support graduate students so that I can direct them in a systematic undertaking to revitalize the curriculum in operating systems and computer networks. I strongly believe that this can be accomplished in three years if a concerted effort is made, but because of the time requirements it cannot be undertaken without significant support.

4 Potential Impact and Significance

Many students find the material in a traditional operating systems or networks course abstract and dry. The animation projects will make them seem more concrete and will stimulate the students to explore possibilities in a virtual laboratory setting.

I already have an extensive set of projects that covers about half of the operating systems curriculum. These projects, developed as a result of the equipment from my first ILI grant, transformed the operating systems curriculum at UTSA into one that is both challenging and exciting. With the publication of my book, *Practical UNIX Programming, a Guide to Concurrency, Communication, and Multithreading* [28], this curriculum is spreading to other universities. Although the book has only been out for about a year, it has been used at such places as MIT, Stanford and USC.

These projects addressed only about half of the material in an operating systems course and it is expected that the projects developed under this grant will have a similar effect on the rest of the course. The networks course will be similarly transformed.

This project has significance not only for its innovative content, but also for its use of visual technology and a networked-based model of learning. The operating systems and networks classes are ideal places to begin the infusion of such technology because the students are relatively sophisticated at this point in their studies, and the mechanics of getting things to work can be incorporated naturally into the curriculum. As part of the dissemination, this work the will be available for running over the World Wide Web.

The use of interactive animation is expected to stimulate the creativity of the students while enhancing their learning. Since the programs run over the network with a standard web browser, this project will have a positive impact on nontraditional students, many of whom are single parents, particularly women trying to raise children and balance work commitments while attending college.

As a byproduct, the interactive animation framework used to produce the animations will be made available and will allow educators to extend this work to other areas.

5 Procedures and Methods

5.1 Investigator Experience and Philosophy

I have been teaching operating systems and networks for about a decade, both at the undergraduate and graduate level. In the 1990 NSF-ILI equipment grant entitled "Concurrency Experiments in a UNIX Environment," my coinvestigators and I set up a laboratory of workstations and redesigned the undergraduate operating systems curriculum to allow students to experiment with the principles of concurrency in UNIX without having to write large, extensive programs. The projects are described in a book [28], the table of contents of which appears in an appendix to this proposal.

My philosophy of teaching these courses is based on the understanding that most students do not fully grasp an abstract concept unless they can experience it first-hand. The traditional "hands-on" approach to operating systems requires that students implement a simple operating system or modify an existing one to add functionality. While this hands-on approach provides an in-depth understanding of basic operating system design, it is problematic when used at the average university. Instructors spend considerable class time covering implementation details — leaving less time to cover a full range of topics. The hands-on approach usually requires a dedicated laboratory and substantial support staff, since kernel bugs often result in system crashes rather than just program crashes. Also, the traditional hands-on approach usually does not provide practical programming experience with advanced synchronization and communication constructs. An alternative to the hands-on approach is a theoretical presentation. Such a "hands-off" course covers more material but does not give the student an in-depth understanding of the concepts in practice.

5.1.1 Hands-On Operating Systems

Our approach is different from either the traditional hands-on or hands-off approaches. Students are required to solve realistic problems that illustrate operating systems concepts. Ordinarily this would be a time-consuming task, so we provide students with the tools necessary to make these tasks manageable. In most cases students only need to write the few hundred lines of code that are key to the concepts being presented.

For example, an important topic in the operating systems course is the client-server model of communication. Programs that implement such a model typically include system calls to do network communication. If such a problem were posed in a traditional way, the students would first have to learn how to use sockets or TLI. The parameters for the system calls to establish connections are structures with many fields that need to be precisely set up. If any of the fields is incorrect, the connection is not made and little indication is given of the reason for the failure. Mastering these details distracts the programmer from concentrating on the key concepts of the client-server model.

To address these difficulties in the undergraduate operating systems course, we introduced a simplified communication paradigm called UICI (Universal Internet Communication Interface). UICI is similar in structure to sockets and TLI, but the parameters required to set up the network communication are simpler. Since the students have access to the source code of the UICI implementation, they can later solve the problem directly using sockets or TLI after they understand the basic concepts. Once they have a working program, students can use it to explore connectionless versus connection-oriented protocols, blocking versus non-blocking I/O, asynchronous I/O and signals, multiple independent processes and threads.

5.1.2 Hands-On Networks

Two years ago when teaching a course in networks using the standard text by Tanenbaum [39], I required the students to simulate some of the data link layer protocols described in the text. I provided the routines that implemented the multiple timers (in fact, the timers were implemented by the operating system class), the asynchronous I/O, and the signals so that the students could concentrate on the protocols themselves. The students uncovered a hidden problem in one of the protocols that the instructor (and Tanenbaum) had not realized was present.

After the course was over, a small group of students working on an independent study project animated the best of these implementations using the XTANGO animation system [35]. The animation showed the various queues, the send and receive windows, frames moving from sender and receiver, frames lost and retransmissions due to errors. While it would not be feasible for students in the class to implement such an animation from scratch and still concentrate on the fundamentals of the course material, an instructor can facilitate the process by providing a library of animation calls specific to the problem at hand. The students can then make simple function calls such as

```
initialize_animation();
```

to draw the initial screen and show the empty queues. To show a frame moving from source to destination they could simply call:

```
move_frame(sequence_number);
```

This approach has several advantages. On the one hand it makes the programs more interesting to work on, and on the other it makes them simpler to debug since the students can see what their programs are doing. However, it appears that the same learning could take place by having the students use an interactive simulation of the network protocol, without requiring as much of a time commitment from the student. The students could use the time to explore additional aspects of the network protocols.

It is important that the simulations require active participation of the students. Students who passively watch a simulation or lecture demonstration may not have their minds focused on the problem at hand. An interactive simulation would require the students to decide how the protocol parameters should be modified to produce a desired effect. This would require a level of understanding not usually obtained by watching passively.

Many other network simulations are amenable to this type of simulation. These include Aloha, several versions of CSMA/CD such as Ethernet, contention free protocols such as BRAM (Broadcast Recognition Access Method) and MLMA (Multi-Level Multi-Access Protocol), FDDI and ATM. Several topics in network routing can also be animated including flooding, random walk, sink trees and reverse path forwarding.

5.1.3 Other Experiences with Visualization in the Computer Science Curriculum

I have also introduced instructor-supported animation in the computer organization course [25]. In the section on microprogramming the students write a simulator for a simple microprogrammed machine. Most of the code for such a simulator involves handling user input and displaying the state of the machine to be simulated. I wrote some routines with simple interfaces to allow the students to animate their simulations. The animation eliminated the need for students to worry about displaying the state of the machine after each cycle. Almost all students were able to write the complete simulator partly because the animation allowed them to see exactly what the simulator was doing at each cycle. Their final programs looked very professional and gave them a sense of accomplishment that students rarely get when completing a course project. This semester the course is being taught be a colleague and the completed simulator is being used as a tool by the students to aid in understanding microprogramming.

5.1.4 Experience with Auralization

In addition to using animation techniques, I have experimented with using sound either with or instead of animation. Researchers have found [8, 9, 10, 11, 31] that sound can be used to enhance the understanding of data and program execution. Sound is particularly useful when several activities are going on in parallel [11]. The use of sound as a visualization aid is called *auralization*. Sound has been used "after the fact" to illuminate relationships that are already understood. It is important that tools for using sound be developed so that sound can easily be incorporated into animations. While some reports of auralization tools [34] have appeared, these typically have a narrow range of applicability. I have incorporated a sound interface into the XTANGO animation package [26, 27] and used it in conjunction with the microprogramming animation described above. Sound is useful for directing the attention of the user and for highlighting the importance of certain events. I plan to continue the use of sound in this curriculum development effort.

I have also incorporated auralization into my research. I have experimented with sound to enhance perception of memory reference patterns generated by programs run on supercomputers. It is possible to hear certain relationships among properties of memory references generated by the TRFD Perfect Benchmark that were found by other methods [24].

5.1.5 Network Monitoring and Animation

Two years ago I was involved in a project with a master's student to develop a STREAMS device driver for Sun Solaris 2.4 [2]. The driver was a multiplexor that sat between an IP driver and a TCP module. It copied TCP packet headers into a FIFO that was read by an animation and monitoring program.

This past year I developed a simulator for the data link layer protocols described in Tanenbaum's texts [39, 40]. This simulator is described in Section 5.5 along with proposed refinements. I used this simulator in my undergraduate networks course this semester with good results.

5.2 The Platform of Choice

This project could not have been proposed a year ago. The enabling technology for this work is Java [5], a programming language that has swept into the forefront with unprecedented speed. The use of Java has been accompanied by hype and extraordinary claims, but putting these aside, Java has the following properties that make it ideal for the proposed project.

- It has been designed from the ground up to allow programmers to incorporate object-oriented design.
- It is available on virtually every computer system from inexpensive PCs to workstations and mainframes.
- It comes with a graphical user interface that is standard across platforms.
- It is an interpreted language so it does not have to be compiled for a particular architecture.
- The interpreter is already installed on millions of machines, and is built into all of the standard web browsers.
- The source code (actually an intermediate form of the code) can be stored and maintained in a central location. This code is grabbed each time it is loaded onto a remote system so the latest version is always used. Virtually any machine connected to the Internet can run this code.

I have shown that the Java environment is suitable for the planned application by developing and animation tool called JOTSA [29] and implementing a sample application described in Section 5.5. JOTSA stands for Java On Time Synchronous Animation.

One of problems with most other animation packages is that they are designed for a given architecture or operating system that is not universally available. I found this problem particularly limiting in the microprogramming simulator described earlier. It was based on XTANGO which required an X GUI and had to be recompiled for each hardware-operating system combination. Basing such a package on Java solves this problem.

Another problem is the speed at which animations run. Although some systems allow the user to slow down the animation, the rate at which an animation runs depends on the speed of the hardware. JOTSA solves this problem by using real time to specify the time it takes to move an object along a path. I call this feature *on time animation*. The user can still adjust the rate of movement, but it is by stretching or shrinking real time. JOTSA animations run at the same speed on slow, fast and very fast machines. If the hardware is too slow to keep up with the animation, JOTSA automatically updates the screen less often, but the animation proceeds at the correct rate. Another feature of JOTSA is its synchronous nature. If several objects are moving, each frame shows the correct relative positions of these objects. JOTSA also adapts to its environment and automatically tunes its performance when the load on the machine changes.

5.3 Related Work by Others

Research from the areas of program and algorithm animation, visualization for program monitoring and debugging, and integrated multimedia systems are relevant to this proposal, as well as an exploding literature on the use of the World Wide Web.

Program and algorithm animation involves the mapping of a program into a dynamically evolving graphical representation in order to clarify structure, control flow and data [3, 30]. Program and algorithm animation are closely related and can often be done in the same system, but algorithm animation usually refers to representations of an abstraction of the approach used to solve the problem.

Algorithm animation has been found effective in the computer science curriculum for helping students synthesize information. Studies have found that interaction, experimentation and active design are crucial to the learning process [4, 19]. Rasala et al. describe a technique for incorporating animation into beginning courses [22]. Students first execute animations to enhance their general understanding of an algorithm. They then run a program to generate different data sets to animate. Finally they write their own programs and analyze the resulting data in a spreadsheet. Naps and Swander [21] have developed an object library for visualizing common data structures such as stacks, queues, binary trees and graphs. Students can make simple calls in their own programs to dynamically display the evolution of the program data structures during execution.

The algorithm animation work most directly related to the current proposal is that of Hartley [12, 13]. He describes several projects for illustrating operating system concepts using the programming language SR and the XTANGO animation system [35]. He provides many interesting variations on classical synchronization problems which students can program. He also has written an SR interface for XTANGO. From my point of view, the main drawback of this work is the use of SR rather than a more mainstream language integrated into a standard operating system. The SR XTANGO interface is low-level in the sense that students have access to functions to draw and move objects such as circles and lines. They then construct their animation from these basic building blocks.

While algorithm animation works at an abstract level, program visualization for monitoring and debugging often shows program execution at a detailed level. Visual debuggers are now commonly available for most languages. DYNALAB [7] is a software system that can be used by undergraduates to monitor Pascal or Ada programs and trace their execution. Object-oriented programs are particularly amenable to visualization because of their organization [23, 42], and program visualization tools are becoming more widely available in industry.

Visualization of operating systems and networks programs have much in common with parallel program visualization because of the concurrent and asynchronous nature of program events [6]. PARADE is a general research environment [37] based on the POLKA parallel animation language [36] for animating the execution of parallel programs. Conch is a library of default program views developed under this system for monitoring parallel program performance. Zhao and Stasko [44] have also used POLKA as a platform for monitoring threaded programs. They replaced the POSIX threads library calls with macros to instrument a program for visualization. Their library is called *Gthreads* and runs on the KSR supercomputers, but the same approach can be used in other situations.

Another aspect of this project is the exploration of new technology for teaching. Multimedia, collaborative systems and the World Wide Web present opportunities for the development of new models for instruction [33]. The use of multimedia has been found to be effective when used in an active learning environment [14, 32, 43]. Many research groups have developed systems for multimedia collaboration and visualization, for example, Shastra developed at Purdue University [1] and Viscomp developed at the University of California at San Diego [18].

A number of experimental systems such as MEDUSA (Multi-participant Engagement through Desktop Unimedia Systems and Applications) from the Interactive Media Laboratory at Georgia Tech [20] provide the unification of multiple media including video, audio and shared windows on a single system.

New developments with the World Wide Web, particularly the Java programming language [5] hold a great deal of promise for development of new educational tools. Many animation projects have been done in Java over the last year. Some of these are described in the over 150 books on Java that have been published or are now in press [41]. Others are described on various web sites [15, 17, 16].

5.4 Plan of Execution and Timetable

This proposal has three major thrusts:

1. Continue development of the Java On Time Synchronous Animation (JOTSA) package which will be the platform for the development of the animations. Concentration will be on a framework for interactive animations.

2. Develop operating systems interactive animations to complete the work already started on a hands-in operating systems curriculum.

3. Develop a curriculum and a set of animations for a hands-on course in computer networks.

The base of JOTSA is complete. Creating a framework for interactive animation will be the most challenging part of the project, but it is critical to productivity on the rest of the project.

My book [28] provides a strong starting point for the development of a laboratory-based curriculum in operating systems. I will develop interactive animations for the operating systems topics not already represented in this book. These include memory management, process scheduling, security and I/O. A guide for instructors will be developed showing how to incorporate these animations into an operating systems curriculum. Most work of this should be completed and put on the web for use by others by the end of the second year.

The standard undergraduate computer networks course suffers from the same problems as the "hands off" operating systems course. There are tremendous opportunities for developing projects with a strong visual component. In the second and third years of the grant I will concentrate on developing a significant number interactive animations suitable for undergraduate computer networks course. I will incorporate the visual components developed in my operating systems work into these projects.

This sections lists several of animation projects that would fit with the plan presented here. The list is far from complete and is only a representative sample. Each animation would include an interactive component described below along with the concepts to be illustrated.

1. **Virtual Rings** — create a ring of processes that communicate via pipes.

   Concepts: process creation and termination, shared resources, simple communication, synchronization via token-passing.

   Interactions: The basic operation would trace through how a ring of processes can be created using fork, dup and pipe. A simple version of this animation has already been implemented. See `http://vip.cs.utsa.edu/pup/forkandpipe.html`. In the full implementation the user would be able control the sequence of pipe, fork, dup and close calls, and the animation would show how the processes are connected. Students could try to implement a bidirectional ring, a mesh, or a hypercube of processes. Once a configuration is generated, the animation could save the sequence of operations so that they could be used to guide the student into writing code implementing the same configuration.

Extensions The animation could be extended to illustrate leader election, anonymous rings, parallel ring algorithms or ring network protocols.

2. **Timers** — show how multiple software timers can be implemented from a single underlying hardware or operating system software timer.

   Concepts: clocks, timers, interval timers, signal handling, critical sections and real time processing.

   Interactions: The user would choose the number of timers and choose one of a small number of implementation methods. The user could devise a scenario of a sequence of timer start and duration times. The signal handler latency could be adjusted to see how it affects timer operation.

3. **Shells** — illustrate a shell with job control.

   Concepts: process control, process groups, foreground and background processes, signal handling.

   Interaction: The user could create sequences of pipes of processes and move processes from foreground to background. The user can see how the process group IDs, the session IDs, and the controlling terminals change as processes are created and moved.

4. **Not Too Parallel Virtual Machine** — animate a simplified version of PVM (the Parallel Virtual Machine) [38].

   Concepts: monitoring of multiple file descriptors, barriers.

   Interactions: The user can set the number of hosts, the number of tasks per host, task operations and barriers.

5. **Tuple Space** — animate the operation of `Richard`, a modified version of the Linda tuple space model.

   Concepts: tuple space model, remote procedure calls (RPCs), stateless servers.

   Interactions: The user can create active tuples representing processes and examine the data structures generated.

6. **Dining Philosophers** — animate the Dining Philosophers problem implemented using monitors.

   Concepts: process synchronization and monitors.

   Interactions: The user can choose the number of philosophers and cause some to become hungry or to stop eating. The visualization would show the state of the code for each philosopher plus the shared monitor.

7. **Process Scheduling** — animate some standard process scheduling algorithms such as round robin, shortest job first and priority queues.

   Concepts: process scheduling algorithms

   Interactions: The user could choose the number of processes and the running time for each process as well as I/O times. The animation would keep track of various statistics such as average waiting time.

8. **Memory Management** — illustrate partitioning, swapping and paging.

   Concepts: memory management under multiprogramming

   Interactions: The user could choose a mix of job sizes, and the animation can illustrate where in memory each process resides for each type of memory management scheme.

9. **Virtual memory address calculations** — show how virtual addresses are converted to physical addresses

   Concepts: paging, translation lookaside buffer

   Interactions: The user could choose a page size, the number of levels of page tables, the size of the TLB and a sequence of memory references. The animation would show how physical addresses are calculated from virtual addresses.

10. **Page Replacement Algorithms** — animate FIFO, LRU, Not Least Recently Used, Second Chance and Clock algorithms.

    Concepts: page replacement, Belady's anomaly, working set

    Interactions: The user could chose a sequence of page references and the animation could show which pages are in memory and keep track of page fault rates.

11. **Aloha** — illustrate pure and slotted Aloha for both the finite and infinite case.

    Concepts: contention protocols, effect of transient behavior

    Interactions: The user could design various loading conditions and see how transients can cause the system to transition from underloaded to overloaded. Experiments could be done to determine how many stations are necessary for the infinite number of stations approximation to be valid.

12. **IEEE 802 Protocols** — show how these protocols handle contention.

    Concepts: CSMA/CD, Ethernet, token bus, token ring

    Interactions: The user could set various parameters of the system including the number of stations, the load generated by each station, and the bandwidth. Statistics would be generated allowing the users to compare the protocols.

13. **Network Switching** — show the behavior of different network switching models.

    Concepts: circuit switching, packet switching, message switching

    Interactions: Users could vary the circuit setup time, packet and message sizes, routing delays, and bandwidth and perform experiments to compare these methods.

14. **Routing Algorithms** — show the behavior of various concepts related to routing algorithms.

    Concepts: adaptive vs. nonadaptive algorithms, hierarchical routing, optimality, sink trees, shortest paths, datagrams, virtual circuits

Interactions: The user could interact in a number of ways. For example, to illustrate a shortest path algorithm, the user could design a graph or choose one of a number of predefined ones. The algorithm could be illustrated one step at a time or the user could be quizzed as to which arc would be added next.

5.5 An Example Animation: Network Protocols

I have implemented an animation of a collection of network protocols from Tanenbaum's Computer Networks book [40]. This book contains a discussion of a sequence of six progressively more sophisticated data link layer protocols. The animation was implemented in JOTSA to test its capabilities and to enhance the unit on data link layer protocols in the course I am currently teaching to 37 seniors in computer science.

Currently, the animation demonstrates the first 5 of the 6 protocols. When the animation is started, the sender and the receiver are displayed along with a number of buttons for control. One of the 5 protocols can be chosen. Figure 1 shows the initial display for protocol 5, a sliding window protocol. The windows of the sender and receiver are shown as well as statistics for the sender and receiver. The user can pull up a control panel as shown in Figure 2 and adjust various parameters such as the error rates and timeout values. Individual frames or a group of frames can be sent.

Each of the 5 protocols has up to 4 built-in scenarios. Each scenario corresponds to a sequence of events that illustrates a feature of the protocol. A commentary window can be pulled up which explains the scenario and follows it as it progresses, highlighting a description of the current state.

Figure 3 shows the commentary window for the first scenario of protocol 5. The upper box gives a brief description of the protocol, and the next one gives a description of the scenario. The two bottom boxes give the sequence of events for the sender and the receiver. As the scenario is played out, lines in these boxes are highlighted showing the progress of the scenario. The animation also supports the use of sound. A running aural commentary describes what is happening and what will happen next. Aural commentaries are particularly useful because students can process audio and visual data in parallel.

In the scenario shown, 7 frames are sent but one frame gets lost. Since the receiver's window has size 1 in this protocol, the receiver throws away the frames it receives out of order and all subsequent frames must be resent. Figure 4 shows the main screen as the frames are moving from sender to receiver and frame number 3 is being lost, moving into the bit bucket.

In its current form, I would classify the protocol animation as mostly-passive. The student can choose which protocol to examine and which scenario to demonstrate and then start it going and watch the results. There is some limited active capability, in that the student can set an error rate for both data and acknowledgments, modify the processing overhead and modify the sender's timeout. The student can devise simple experiments to answer questions such as "What would happen if the first frame is delivered successfully but the timeout is set too small."

To turn this into a user driven interactive learning tool, the student would have to be able to invent scenarios like the ones that I have coded into the animation. The instructor could then pose a problem like the following:

> We saw that in Protocol 4, a certain sequence of events could cause each frame to be sent twice even though no additional errors occur. Start with this configuration and see if you can cause a single error that will make each frame be sent three times. How could the protocol need to be modified to inhibit this behavior?

The animation would present the user with a menu in which the history of each frame could be specified. For example the user would specify whether the frame would be received correctly, whether the acknowledgment would be received correctly, and how long the receiver would delay in acknowledging the frame. Similar information could be set for each retransmission.

At any time the user could stop the simulation and change parameters. The state of the simulation and the sequence of events that brought the simulation to that state could be saved or restored.

The modifications to the animation needed to accomplish this are relatively minor, compared to the work already done, but they would be time consuming. This work would could be carried out by the graduate students supported by this grant.

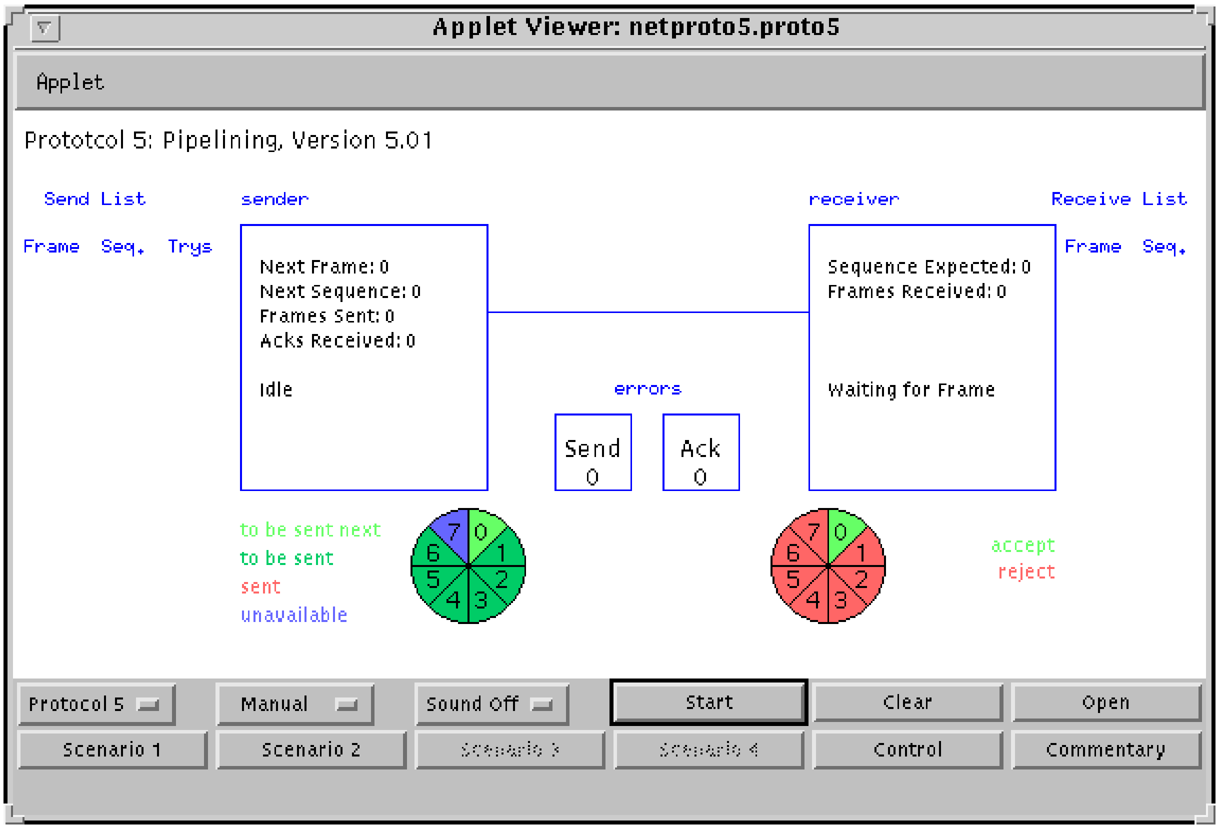

Figure 1: The initial display after protocol 5 is chosen.

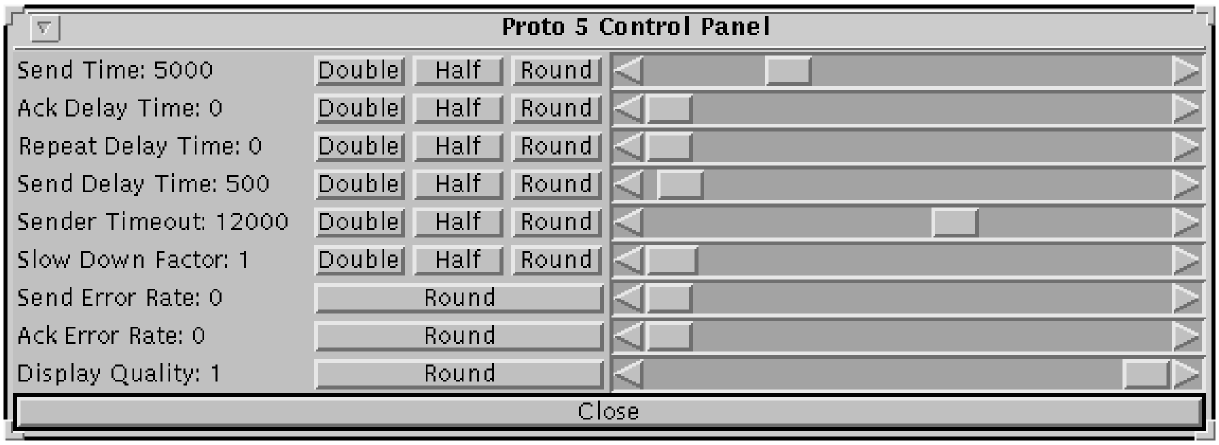

Figure 2: The control panel.

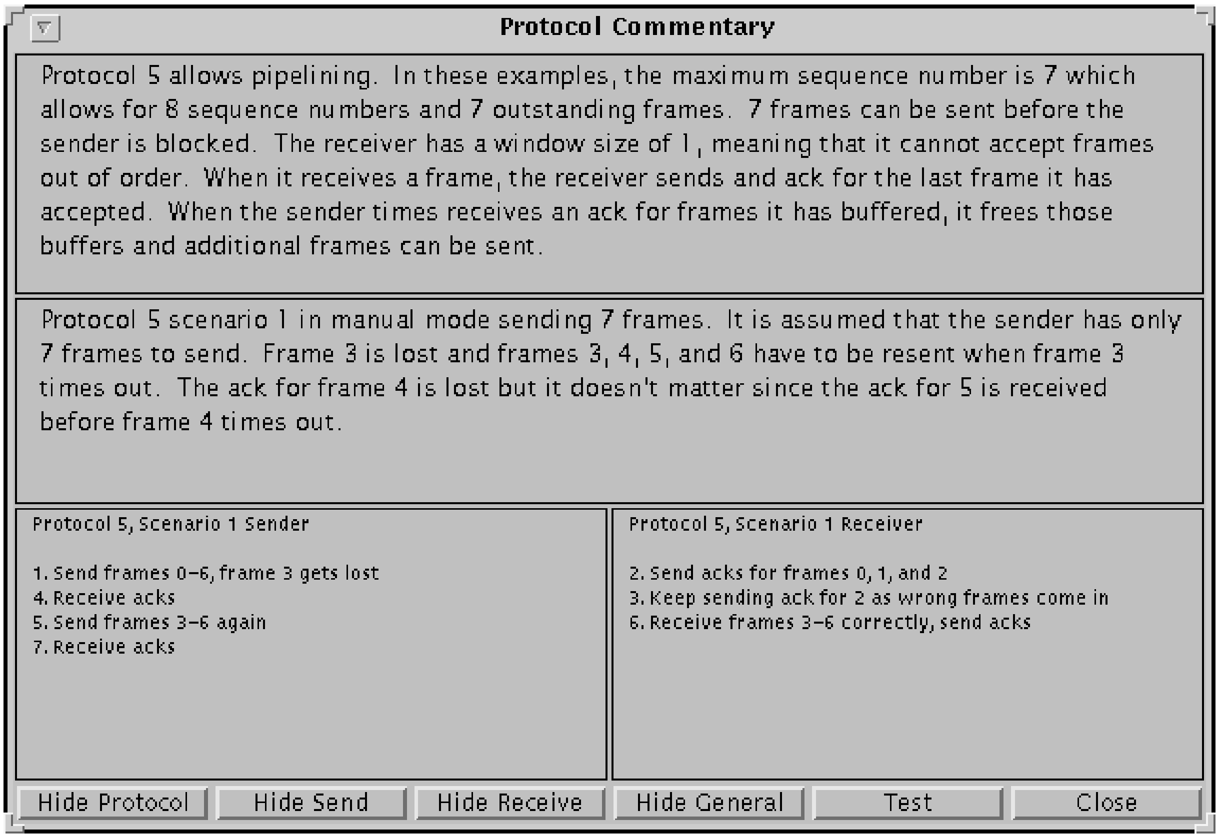

Figure 3: The commentary for scenario 1 of protocol 5.

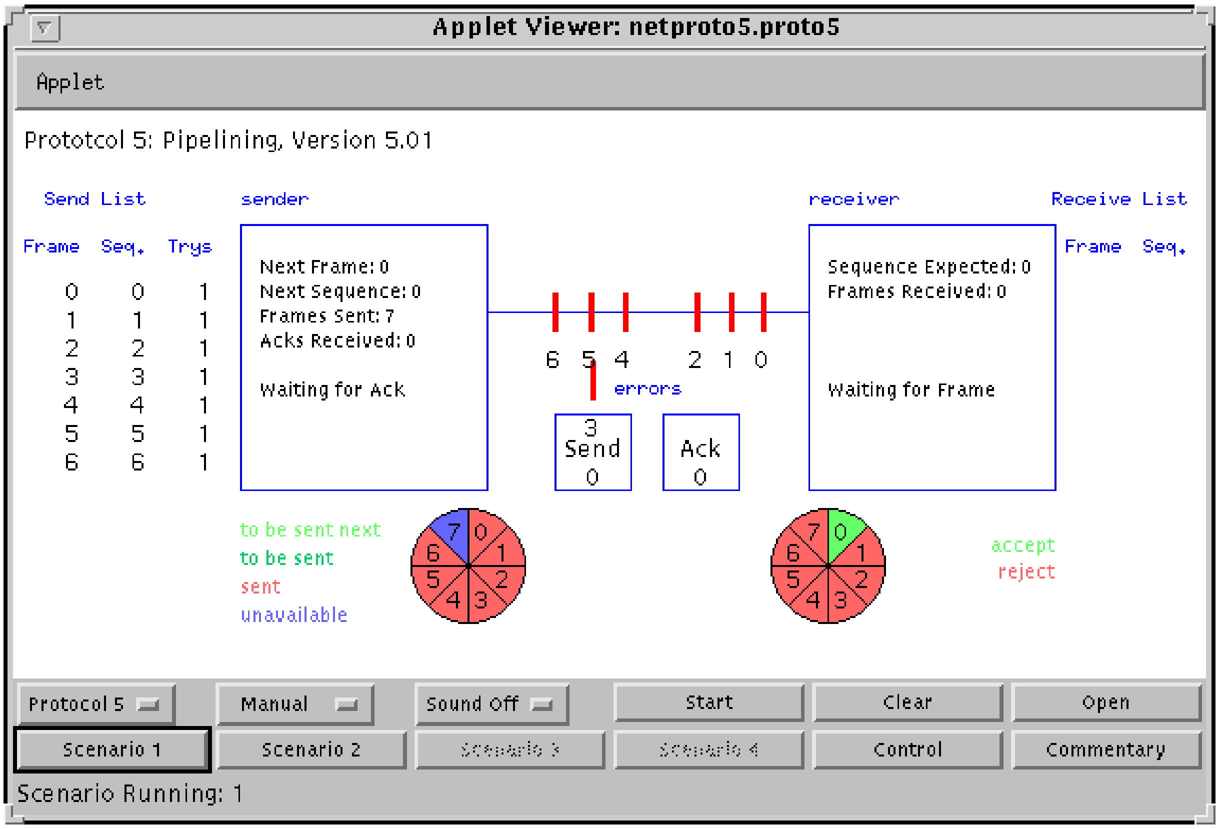

Figure 4: A snapshop of the display while scenario 1 is in progress.

5.6 Integration into the Institution

I regularly teach the undergraduate operating systems class which is offered every semester and the computer networks class which is usually offered once a year. Curriculum advances will be integrated immediately into these classes. The other faculty member who regularly teaches these courses has agreed to use these projects as well. I will present a yearly seminar at UTSA to discuss this work and the results so that other faculty will be aware of the progress and opportunities. Once the material has been developed, no continuing cost is associated with using these projects.

5.7 Institutional Profile

The University of Texas at San Antonio was founded in 1969 and has rapidly grown into a comprehensive metropolitan institution serving over 17,500 students. The institution is 40% minority and has a significant enrollment of nontraditional students. A separate Division of Computer Science was formed in October 1995. The computer science program has about 400 undergraduate majors and 65 masters students. A doctoral program in computer science was recently inaugurated and the first six students were admitted in September 1995. The program has thirteen full-time faculty members. Teaching assistants handle certain service courses for nonmajors. They also teach the laboratories associated with the lower division courses for majors. The undergraduate computer science program requires an introductory programming course, a data structures course, two semesters of computer organization, analysis of algorithms, operating systems, computer architecture, programming languages, software engineering, systems programming, data base management and discrete mathematics. Students are also required to take three semesters of calculus, linear algebra, two semesters of probability and statistics, a laboratory science and technical writing in addition to the core curriculum required of all students. Computer science electives include computer networks, artificial intelligence, graphics, automata and compilers.

5.8 Expected Impact on Students

The innovative curriculum will better prepare students for a rapidly evolving job market. Many of the students from UTSA's computer science program prefer to be systems programmers and developers at small companies rather than to do software maintenance in IS departments at large companies. The approaches outlined in this proposal are more likely to encourage their creativity and their interest in remaining on the front line of technological developments. We want to produce innovators, not legacy programmers.

One of the advantages of this work is that the projects can be run on any type of computer running a browser. This type of curriculum will have a major impact on women who often do not wish to travel to the university in the evenings because of family obligations. Being able to work at home during these times would give them more time to spend on their studies. Similarly, the ability to work at home could be helpful to those with disabilities.

6 Evaluation

Many students who have taken my operating systems class have contacted me after they graduated and have indicated how valuable their operating systems class was to them. With this grant we will try to formalize the feedback mechanism. I will administer student profile questionnaires to all students who participate in my operating systems and networks courses. I will then administer follow-up questionnaires to students one, two and three years after they have taken the course. This will require tracking these students after they graduate since most students enroll in these courses during their last three semesters of university study.

I will be presenting the results of my work at conferences and expect to have it evaluated by the usual peer review associated with conference submissions. I will also solicit criticism when the work is presented.

It is hard to completely separate evaluation and dissemination. The ultimate test of this project lies in its impact beyond UTSA. My previous NSF ILI grant resulted in the publication of a book which was published last year. This book is being used at such places as MIT, Stanford and USC. I have set up a web site for this book and a link will be included to a homepage for this project. I will track access to this web site. The homepage will request users of the software to fill out a short (3-line) on-line form giving their name and email address. These people will be contacted and a more detailed survey will be performed.

7 Results and Dissemination

An extensive curriculum will be developed and made available on the World Wide Web. Included in this curriculum will be source code, class notes, implementation notes, and transparencies. Each of the animations will be able to be run from any machine having a browser with access to the Internet. The work will be presented at standard conferences such as SIGCSE.

References

[1] A. Anupam and C. L. Bajaj, “Shastra: Multimedia collaborative design environment,” *IEEE MultiMedia* (1)2 pp. 39–49, 1994.

[2] A. D. August, *TCPMON: A Passive and Secure Sound Network Monitoring System for TCP Applications*, UTSA Master’s Thesis, 1995.

[3] M. H. Brown and J. Hershberger, “Color and sound in algorithm animation,” *Computer*, 25(12) pp. 52–63, 1992.

[4] C. Brown, H. J. Fell, V. K. Proulx, and R. Rasala, “Using visual feedback and model programs in introductory computer science,” *Journal of Computing in Higher Education*, 4(1) pp. 3–26, 1992.

[5] B. Carlson, “ A jolt of Java could shake up the computing community,” *Computer*, 28(11) pp. 81–82, 1995.

[6] T. L. Casavant (Ed.) Special issue on tools and methods for visualization of parallel systems and computation,” *Journal of Parallel and Distributed Computing*, 18(2), June 1993.

[7] M. R. Birch, C. M. Boroni, Grances W. Goosey, S. D. Patten, D. K. Poole, G. M. Pratt and R. J. Ross, “DYNALAB A dynamic computer science laboratory infrastructure featuring program animation,” *Proc. 26th SIGCSE Technical Symposium on Computer Science Education*, pp. 29–33, 1995.

[8] J. M. Francioni, J. A. Jackson and L. Albright, “The sounds of parallel programs,” *Proc. Sixth Distributed Memory Conf.*, pp. 570–577, 1991.

[9] J. M. Francioni and J. A. Jackson, “Aural signatures of parallel programs,” *Proc. 25th Hawaii International Conf on System Sciences*, pp. 218–229, 1992.

[10] J. M. Francioni and D. T. Rover, “Visual-aural representations of performance for a scalable application programs,” *Proc. Scalable High Performance Computing Conf, SHPCC-92*, pp. 433–430, 1992.

[11] J. M. Francioni and J. A. Jackson, “Breaking the silence: auralization of parallel program behavior,” *J. Parallel and Distributed Computing*, 18 pp. 181–194, 1993.

[12] S. J. Hartley, “Animating operating systems algorithms with XTANGO,” *Proc. 25th SIGCSE Technical Symposium on Computer Science Education*, pp. 344–348, 1994.

[13] S. J. Hartley, *Operating Systems Programming, the SR Programming Language,* Oxford University Press, 1995.

[14] E. A. Isaacs and J. C. Tang, “What video can and cannot do for collaboration: a case study,” *Multimedia Systems*, 2 pp. 63–73, 1994.

[15] Java World Wide Web: `http://java.sun.com`.

[16] Java Report Web Site: `http://www.sigs.com/jro/`.

[17] Java Gamelan Web Site: `http://www.gamelan.com/`.

[18] P. Kelly and S. Moezzi, "Visual computing laboratory," *IEEE Multimedia*, 2(1) pp. 94–95, 1995

[19] A. W. Lawrence, A. N. Badre, and J. T. Stasko, "Empirically evaluating the use of animations to teach algorithms," *Georgia Tech Technical Report GIT-GVU-94-07*, 1994.

[20] World Wide Web: `http://www.gatech.edu`.

[21] T. L. Naps and B. Swander, "An object-oriented approach to algorithm visualization — easy, extensible, and dynamic," *Proc. 25th SIGCSE Technical Symposium on Computer Science Education*, pp. 46–50, 1994.

[22] R. Rasala, V. K. Proulx, H. J. Fell, "From animation to analysis in introductory computer science," *Proc. 25th SIGCSE Technical Symp. on Computer Science Education*, pp. 61–65, 1994.

[23] S. P. Reiss, "Interacting with the FIELD environment," *Software–Practice and Experience*, 20(S-1), June 1990.

[24] K. A. Robbins and S. Robbins, "Relationship between average and real memory behavior," *J. Supercomputing*, pp. 209–232, 1994.

[25] S. Robbins, "A microprogramming animation," *University of Texas at San Antonio Technical Report CS-95-10*, 1995. Submitted to *IEEE Trans. on Education*.

[26] S. Robbins, "Adding sound to the XTANGO animator," *University of Texas at San Antonio Technical Report CS-95-13*, 1995.

[27] S. Robbins, "Controlling the Korg X5DR synthesizer from a UNIX program," *University of Texas at San Antonio Technical Report CS-95-12*, 1995.

[28] K. A. Robbins and S. Robbins *Practical Unix Programming: A Guide to Concurrency, Communication, and Multithreading* Prentice Hall, 1996.

[29] S. Robbins, "JOTSA: Java On Time Synchronous Animation," *University of Texas at San Antonio Technical Report CS-96-13*, 1996.

[30] G.-C. Roman and K. C. Cox, "A taxonomy of program visualization systems," *Computer*, 26(12) pp. 11–24, 1993.

[31] C. Scaletti and A. B. Craig, "Using sound to extract meaning from complex data," *SPIE*, 1459, 1991.

[32] R. C. Schank, "Active learning through multimedia," *IEEE MultiMedia*, 1(1) pp. 69–78, 1994.

[33] B. Shneiderman, M. Alavi, K. Norman and E. Y. Borkowski, "Windows of opportunity in electronic classrooms," *CACM*, 38(11) pp. 19–24, 1995.

[34] D. Sonnerwald, et al, "Infosound: An audio aid to program comprehension," *Proc. 23rd Hawaii Intl. Conf. on System Science* pp. 541–546, 1990.

[35] J. T. Stasko, "Animating algorithms with XTANGO," SIGACT News, 23(2) pp. 67–71, 1992

[36] J. T. Stasko and E. Kraemer, "A methodology for building application-specific visualizations of parallel programs," *J. Parallel and Distr. Computing*, 18(2) pp. 248–264, 1993.

[37] J. T. Stasko, "The PARADE environment for visualizing parallel program executions: A progress report," *Georgia Tech Technical Report GIT-GVU-95-03*, 1995.

[38] V. S. Sunderam, "PVM: A framework for parallel distributed computing," *J. of Concurrency: Practice and Experience*, 2(4) pp. 315–339, 1990.

[39] A. S. Tanenbaum, *Computer Networks*, Prentice Hall, Second Edition, 1989.

[40] A. S. Tanenbaum, *Computer Networks*, Prentice Hall, Third Edition, 1996.

[41] G. Vanderburg, *Tricks of the Java Programming Gurus*, Sams.net, 1996.

[42] N. Wilde and R. Huitt, "Maintenance support for object-oriented programs," *IEEE Trans. on Software Engineering*, 18(12) pp. 1038–1044, 1992.

[43] B. P. Woolf and W. Hall, "Multimedia pedagues," *Computer*, 28(5) pp. 74–80, 1994.

[44] Q. A. Zhao and J. T. Stasko, "Visualizing the execution of threads-based parallel programs," *Georgia Tech Technical Report GIT-GVU-95-01*, 1995.

VITA

STEVEN ROBBINS

GENERAL INFORMATION

Title:	Associate Professor
Address:	Division of Computer Science The University of Texas at San Antonio 6900 North Loop 1604 West San Antonio, TX 78249-0667
Telephone:	512-691-5544
Fax Number:	512-691-4437
Email Address:	srobbins@runner.utsa.edu
WWW:	http://vip.cs.utsa.edu/srobbins
Citizenship:	United States

EDUCATION

1973	Ph. D.	Massachusetts Institute of Technology
1969	S. M.	Massachusetts Institute of Technology
1969	S. B.	Massachusetts Institute of Technology

ACADEMIC EXPERIENCE

1996-present	Professor with Tenure, University of Texas at San Antonio
1979-96	Associate Professor with Tenure, University of Texas at San Antonio
1975-79	Assistant Professor, University of Texas at San Antonio
1973-75	Instructor, Massachusetts Institute of Technology
1973-75 (summers)	Instructor, Project Interphase, Massachusetts Institute of Technology
1969-73	Teaching Assistant, Massachusetts Institute of Technology

RESEARCH INTERESTS

Parallel architectures, distributed applications, high performance computing, visualization, systems, concurrency

PROFESSIONAL AFFILIATIONS

Association for Computing Machinery (ACM)
Institute of Electrical and Electronics Engineers (IEEE)

FIVE PUBLICATIONS RELEVANT TO PROPOSED WORK:

1. "A Microprogramming Animation," *University of Texas at San Antonio Technical Report CS-95-10*, (1995), submitted to the *IEEE Trans on Education*.
2. *Practical Unix Programming: A Guide to Concurrency, Communication, and Multithreading* with K. Robbins, Prentice Hall, November 1995.
3. "The JOTSA Animation Environment," *University of Texas at San Antonio Technical Report CS-9703*, 1997, submitted to the *31st Hawaii Intl. Conf. on Systems Science*.
4. "Blackboards for cooperative problem-solving," with K. Robbins, *Proc. ASEE DELOS Conference*, (1993), pp. 459–465.
5. "Ethercom: A study of audio processes and synchronization," with R. Rybacki and K. Robbins, *Proc. 24-th SIGCSE Technical Symposium on Computer Science Education*, (1993) pp. 218–222.

FIVE ADDITIONAL PUBLICATIONS:

1. "Buffered banks in multiprocessor systems," with K. Robbins, *IEEE Trans on Computers*, vol. 44 (1995) pp. 518-530.
2. "Relationship between average and real memory behavior," with K. Robbins, *J. Supercomputing*, vol. 8 (1994) pp. 209–232.
3. "Dynamic behavior of memory reference streams for the Perfect Club benchmarks," with K. Robbins, *Proc. 1992 Intl. Conf. on Parallel Processing*, vol. I (1992) pp. 48–52.
4. "Bus conflicts for logical memory banks on a Cray Y-MP type processor system," with K. Robbins, *Proc. 1991 Intl. Conf. on Parallel Processing*, vol. I (1991) pp. 21–24.
5. *The Cray X-MP: A Case Study in Pipelined Architecture and Vector Processing*, with K. Robbins, Springer-Verlag Lecture Notes in Computer Science #374, (1989).

GRADUATE ADVISORS:
Irving Segal and David Gordon

OTHER:
No thesis students or post-doctoral scholars supervised.

10 Budget

10.1 Budget Justification

The major budget item in the proposal is for salaries. The principal investigator will receive 2 months of summer support for each of the three years of the grant. In addition, NSF will fund one course release for each semester of the first year of the grant. This is equivalent to 2.25 academic months. I anticipate that there will be considerably more work to be done during the first year as the tools will have to be completed and the students will have to be trained. This support will allow the work to proceed in a timely manner.

Two full-time research assistants, one PhD student and one master's student will be supported for each of the three years of the grant. UTSA has agreed to pay the tuition and fees for the PhD student supported by this grant, a commitment of about $3,000 per year. These students will produce most of the simulations.

Three small pieces of hardware will be purchased with this grant. A scan converter capable of producing video output from a high resolution computer display output will be purchased. One such device currently available is the HyperConverter 1280 made by PC Video Conversion. This allows the display output of a Sun workstation or high resolution PC to be used to create a videotape and will handle resolutions up to 1280 by 1024. Videotapes of the animations will be created and used for dissemination of the results of this grant. This device costs about $4000. A writeable CD ROM drive will be purchased for about $1000. This will allow the production of CD ROMs. Many conferences on visualization now require a videotape or CD ROM with submissions. A multimedia notebook PC for about $3000 will be purchased to aid in presenting this material at conferences. UTSA will share half of the cost of all equipment.

An M&O budget of $2000 per year is anticipated. A color laser printer and a frame accurate editing Super VHS recorder are already available for use with the above scan converter. Also available is a system for editing video. Most of the M&O budget will be for supplies such as videotape, toner and transparencies for use in disseminating the results of this work.

I have also included $2000 per year for software for developing Java code in both a PC and Solaris environment. Preliminary versions of these tools such as the Java Workshop by Sun Microsystems are now becoming available. It is anticipated that by the time the grant starts in September of 1997, many more such products will be available.

A travel budget of $3000 per year will allow the investigator and the graduate assistants to attend one meeting per year, first to establish contacts with others pursuing similar work and later to present the results.

10.2 University Commitment

A laboratory of Sun workstations is available in the Division of Computer Science and space has been set aside in this laboratory for projects in visualization. This lab contains several Sun workstations and a Pentium Pro PC. Hardware and software are available in this lab for video capture and editing of video.

UTSA has agreed to cover the tuition and fees of the PhD student supported by this grant, at about $3,000 per year. Half of the equipment cost ($4,000) will be covered by the university. In addition, UTSA has committed $2,000 in each of the second and third years of the grant to cover maintenance of the equipment which will be used for this project.

Appendix

Practical UNIX Programming: A Guide to Concurrency, Communication and Multithreading

by

Kay. A. Robbins and Steven Robbins

Contents and Preface