INTERACTIVE PARALLEL SIMULATIONS WITH THE *JANE* FRAMEWORK

Kalyan S. Perumalla
Richard M. Fujimoto
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280
Email: {kalyan,fujimoto}@cc.gatech.edu

KEYWORDS

Interactive parallel simulation, collaboration, remote execution, graphical interface, client-server

ABSTRACT

Three important practical issues in fostering the widespread use of parallel simulation technology are the availability of graphical user interfaces, accessibility to parallel computing resources, and facilities for collaborating in the execution of simulations. First, graphical user interfaces must be available to support the various phases of the modeling and simulation process. Secondly, potential users who may not have the necessary parallel computing resources locally must be able to remotely access those resources elsewhere. Finally, collaborative simulations are important because expert modelers are often not parallel simulation experts (and vice versa), and they may be geographically separated from one another. The availability of sophisticated graphical interface development languages and environments, coupled with the advent of universal accessibility via the Internet, makes it easier to address these issues. As part of our projects in high-performance telecommunication network simulation and in distributed federated simulations, we are developing a simulator-neutral interactive simulation framework, called Jane. Jane is intended to help users to remotely and collaboratively interact with parallel simulations over the Internet, and to help them view their own model-specific run-time animations. We document architectural details of the Jane framework, along with the issues and challenges that we faced in designing and implementing interactivity features in the framework, specifically in the context of our optimistic parallel simulator, GTW (Georgia Tech Time Warp), and the telecommunication modeling language, TeD.

1. INTRODUCTION

In both sequential and parallel/distributed simulations, visualization is an invaluable tool for the modelers. In the case of parallel/distributed simulation, additional motivation arises for supporting visualization—model debugging (especially for optimistic simulations) and performance enhancement may become easier with the help of graphical visualization.

Most current commercial sequential simulation packages routinely support sophisticated graphical interfaces for model development, debugging and visualization. The same cannot be said to be true for parallel simulation systems, most of which still are primarily research-oriented. User-friendly graphical interfaces could potentially help in the widespread adoption of parallel simulation systems. Moreover, remote access to parallel computing resources is necessary to facilitate the execution of parallel simulations by those users who do not have local access to expensive parallel computers. However, important issues have to be addressed to make these features possible.

First, an approach is needed by which the interface software must be clearly de-linked from the parallel simulation software. This is because parallel simulation kernels are inherently very complex, and hence it is important to avoid additional complexity that can result due to any tight coupling between the complex kernel/model software and the graphical interface software. Another reason to avoid tight integration is to accommodate legacy simulation software systems, into which it is difficult to incorporate graphical interactions. Moreover, the programming language used for a simulation kernel may not be the most appropriate one for building powerful graphical interfaces. The *client-server* approach appears to be most appropriate for achieving all these goals.

Secondly, usability features have to be addressed bearing in mind the simulation model developer and the model user. Ideally, simulation models must be independent of the particular views used to visualize and control the models, since different users can develop different visualizations of the same model. Also, facilities for instrumenting the simulation models for model-specific visualizations must be natural to use for the modeler, without burdening him with the details of the synchronization protocol (optimistic or conservative) used underneath. Instrumentation and synchronization issues must be carefully solved to adequately address these concerns.

Thirdly, the nature of large-scale model development has to be carefully considered. Many successful modeling and simulation systems are built using a layered approach. For example, a modeling language is used as insulation from the simulator's primitives (e.g. Maisie). Domain-specific frameworks

are built using the modeling language, which are general enough to describe a variety of models in a domain (e.g. wireless networks). Actual user models could merely be instantiations within such frameworks. It becomes important to have an interactive system architecture that ensures that the system continues to function in the presence of errors in any of the layers.

Finally, it is important to recognize that successes in the application of parallel simulation technology to real-life applications requires close cooperation of parallel simulation experts with the modelers who are experts in the given application domain. Hence, it is important for interactive systems to provide facilities for geographically distributed users to collaborate during model development and simulation execution.

These issues are in addition to performance issues of interactive systems, such as maintaining adequate rate of simulation time advance, state saving techniques for interactive optimistic simulations, and the like.

As part of our parallel and distributed simulation projects, we are addressing these issues in our framework for interactive parallel simulations, called Jane. Jane is intended mainly as a general simulator-independent framework for interactive simulations. Into this framework, we have integrated two different simulation systems. The first simulation system is an implementation of the High Level Architecture (HLA) Run Time Infrastructure (RTI) interface (see HLA web page). In fact this simulation system is a class of several federated simulators that are developed using the tool kit called the Run-time Infrastructure Kit (RTI-Kit), one of which implements the HLA RTI. The second simulation system is the Telecommunications Description Language (TeD) for the parallel simulation of large-scale telecommunication networks (the S3 project 1998). The TeD modeling language (see TeD/Jane web page 1998) is used to model the structure and behavior of large-scale telecommunication networks. TeD models can be compiled to any simulator (sequential, conservative or optimistic). Currently we use the Georgia Tech Time Warp (GTW) simulator, which is primarily a research-oriented optimistic parallel simulator developed over the last decade.

Related Work

Graphical interfaces and visualization for parallel simulation models are being developed in some recent efforts. Some parallel simulation systems have started including support for graphical interfaces for model development, debugging and performance visualization (e.g. Maisie/PARSEC). Animations of processor behavior and performance visualization are presented in (Carothers *et al*. 1997)). Performance issues related to interactive parallel simulation have been well studied (Carothers *et al*. 1997; Graham *et al*. 1996; Franks *et al*. 1997). An empirical evaluation of the effectiveness of performance visualization features is presented in (Graham *et al*. 1998). The state saving and performance issues have been studied in (Franks *et al*. 1997) in the context of real-time interactive simulations.

In contrast, our work is focussed on developing an interactive simulation framework that acts as the glue for incorporating the various interactivity features, applying them in real-life parallel simulation projects (e.g. large-scale telecommunication), and testing and improving them continually based on feedback from the users.

In section 2, we present an overview of the Jane client-server architecture. In section 3, we introduce some of the default views supported in Jane, including the simulation and model display/control features. In section 4, we list some practical issues and solutions that arose in implementing the interactive framework of Jane. Finally, we report the current status of Jane along with intended future work. Throughout our discussion, we illustrate the features of Jane with its use in the context of the TeD/GTW system.

2. JANE OVERVIEW

Jane is primarily a graphical user interface system to parallel and distributed simulations. In addition to providing default graphical controls and displays for the core simulation systems in a simulator-neutral fashion, Jane provides robust and extensible mechanisms for users to incorporate their own model-specific views to override or supplement the default views.

Jane is based on the client-server architecture. The server and the clients communicate over the Internet. The server is written in C, while the clients are written in Java (the architecture is capable of supporting others, such as a Tcl/Tk or Visual Basic client). The client supports a set of default graphical controls and views, but also provides support for incorporating arbitrary visualizations of models. For example, the TeD models can be instrumented in a natural way, and the instrumented information is automatically made available to the controls and visualizations on the client end using a *service-user* design pattern. Although we are currently focusing on TeD and RTI-Kit specific controls and displays, the architecture is sufficiently general to incorporate any other application-specific interfaces. Thus, Jane is simulator-neutral, making it reusable across different simulators and models.

Some of the intended capabilities of Jane are:

- Remote control of parallel simulations over the Internet (from a laptop, for example)
- Default graphical views and controls for parallel simulations
- Default graphical views for runtime visualization of modeling system (such as for TeD simulations)
- Easy development of application-specific animations (such as for TeD models)
- Scripting for programmatic control of parallel simulations
- Pre- and post-simulation analysis of multiple simulations

- "Simulation gateway" capability to interface with larger design/decision tools.

Jane is written with the intent to be language neutral, and to be easily extendable for model specific animations. In addition, Jane supports the use of the simulation in a larger design problem (for example, Network Design and Optimization) in which simulation is just a sub-component. It can also be used for programmatic "design of experiments", automatically running multiple sequential or concurrent simulations (for good confidence intervals), and for the analysis of simulation results.

Client-Server Architecture

Jane is based on the client-server architecture, supporting the independent operation of client, server, and simulator processes. Figure 1 depicts a snapshot of a sample configuration of the clients, servers and simulations in operation.

Client 1 is connected to server 1 and server 2, controlling simulation 1 and simulation 2 respectively. Client 2 is also connected to server 1 to control simulation 1 (clients 1 and 2 are collaborating in interacting with simulation 1). Client 3 is controlling simulation 3 via server 2. Users using their graphical interfaces operate clients 1 and 2. Client 3 is in fact a script that is controlling simulation 3.

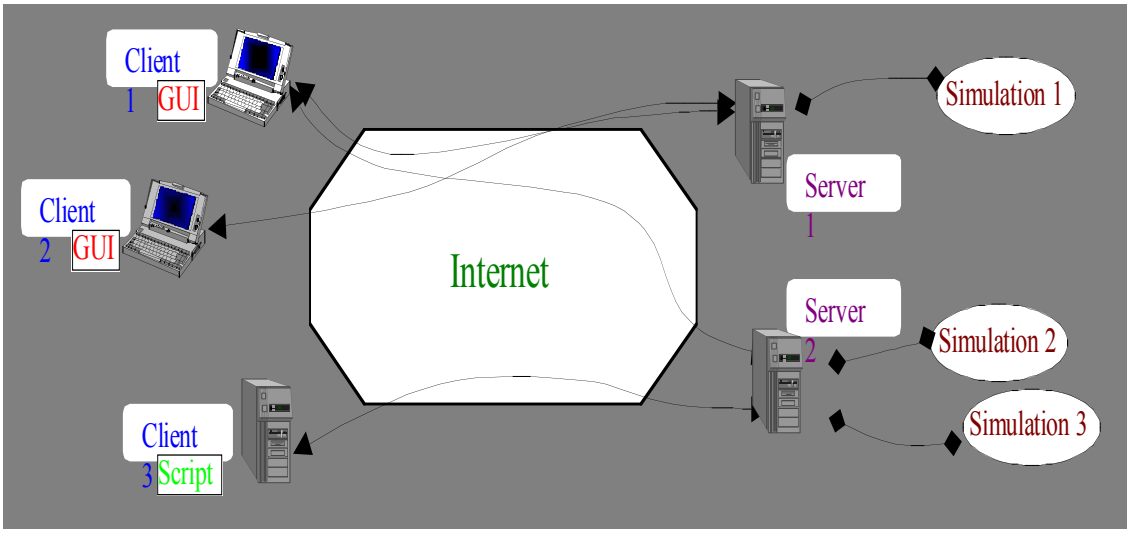

Figure 1: Client—Server Architecture for a simulation session in Jane.

In general, a client can simultaneously connect to several servers, and a server can be actively servicing multiple clients. Each server can spawn and control several simulation runs at the same time. Several collaborating clients may control a single simulation.

Software

The current Jane software consists of

- a server written in C
- a client written in Java
- a model-instrumentation API (Application Program Interface) in C/C++, and
- simulation-control and model-specific visualization API in Java.

Currently, the communication between clients and servers is over the network using TCP/IP (Transmission Control Protocol/Internet Protocol) sockets. The communication between servers and simulations is via Unix pipes – our current implementation hence requires that a simulation be executed on the same machine as its controlling server.

Simulation

The simulator communicates directly with the server. This is done automatically and transparently—the GTW and RTI-Kit applications are unaware of this connection. The Jane client controls the simulations, and the graphical feedback of simulation performance can be monitored at the client. Any GTW or RTI-Kit simulation application can be run in this environment. In particular, since simulations of TeD models are in fact GTW applications, any TeD simulation can be run using Jane. Similarly, since HLA RTI federates are in fact RTI-Kit applications, any HLA federation can be run using Jane.

Models can be instrumented using the instrumentation API, which includes natural primitives for exposing certain constants and variables of the model, and sending instrumentation events to the

environment in order to signal the actions of interest in the model execution.

Server

The server is a daemon process that accepts connection requests from clients and performs operations such as spawning the simulation processes, conveying information between the client and simulation, and taking care of termination of simulations. Since the server is a process separate from the client or simulation, it can survive both simulation crashes as well as client crashes. It can also help the client in terminating a runaway simulation if necessary. The Jane software includes a server written in C.

Client

The client provides a standard set of features to allow for simulation control and monitoring. In addition, the client has an API that allows customization through user objects. The client performs three types of functions:

1. Simulation display and control
2. TeD or RTI-Kit display and control
3. Model-specific display and control.

The client supports TeD-specific and RTI-Kit specific default views, which are automatically generated for any TeD model or RTI-Kit federation.

The Jane software includes a client written as a stand-alone Java program, which supports a graphical user interface and includes a scripting API. Graphical interface is built using the Java Foundation Classes (JFC, formerly known as *Swing*). The client in fact is a library of Java packages, each of which defines an API for a set of interaction operations. The client talks to the server via a TCP/IP socket.

3. SAMPLE JANE SESSION

We give a brief introduction to the Jane environment by illustrating a sample Jane simulation session in the context of a TeD model. Before initiating simulations in the Jane interactive system, the server must be started on the parallel computer. The client is run on the machine from which simulations are to be controlled (say, from a laptop with connection to the Internet). The main client window in a sample simulation session is depicted in figure 2.

The client is connected to the server using the Server-Connect dialog, and the simulation is spawned and started using the Simulation-Start dialog. Simulation progress is controlled using the Pause, Resume, Stop and Kill options. An identification area displays the list of current participants (i.e., other clients) that are collaborating in this session. A chat area allows participants to communicate with one another by text messages, which appear tagged with user's initials in all client displays. Simulation time advances and percentage of completion are displayed in a gauge. Aggregate and individual processor event statistics are automatically collected periodically and continuously displayed in the corresponding windows. The frequency with which statistics are collected can be increased or decreased using the view options depending on user requirement. In the case of distributed simulations (such as RTI-Kit simulations), the output from each process is displayed in a separate text area contained in a tabbed pane that houses all the text areas, thus avoiding the clutter of multiple windows.

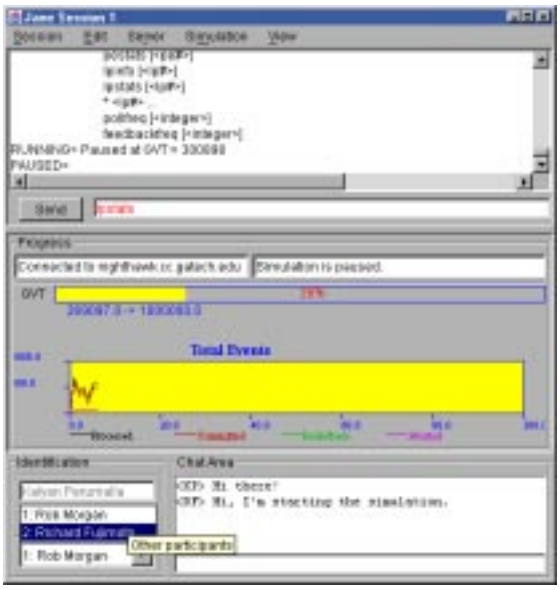

Figure 2: Main Jane client window showing a sample collaborative session

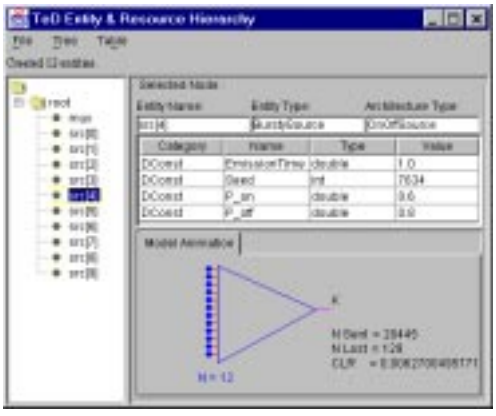

Figure 3: TeD entity hierarchy view

Graphical views of the entities in the TeD models are automatically drawn in the entity hierarchy view that is natural for TeD semantics (see figure 3). The nested entity instances are depicted in the tree structure, while

their corresponding list of constants and variables are displayed beside the tree. Drawing area for model-specific animations is also included. Figure 3 illustrates a model-specific animation, which is a visualization of cell-loss rate in an ATM Multiplexer simulation.

A new client window can be spawned while keeping the current session window, similar to the capability of a web-browser to start a new web browser window. The new copy of the client can establish and maintain a separate connection to another server, independent of any other active sessions.

4. IMPLEMENTATION ISSUES

Several important practical issues arise in supporting remote graphical interaction to parallel simulations such as in Jane. We document some of these issues and our solution approaches below.

Instrumentation

Instrumentation is the process of augmenting the otherwise pure model with additional statements to instruct the runtime system to display model-specific views. It is common to find systems where the model developers themselves develop customized graphical views of their models, thus making their instrumentation view-dependent. On the other hand, it is preferable to have the modelers expose the items and points of interest in a general way, which are de-coupled from graphical views. This makes the instrumentation animation-independent, allowing other users to build different views using the same instrumentation data. In TeD, for example, instrumentation is supported naturally using the same event-sending primitives as used to define the TeD model behavior. While the normal events are sent by an entity over its declared *channels*, the instrumentation events are sent by the entity over a predefined stdout channel (analogous to standard output stream of C/C++ programs). The events sent on stdout channels are captured by the runtime system and submitted to the Jane client. Using a simple API, the users can define their own classes that receive the instrumentation events and augment/override the Jane client display with customized views. This permits the users to change the views at any time without changing the instrumentation.

In figure 3, for example, the entity tree and the table of variables is displayed by default for all TeD models. However, as a customized display for the ATM Multiplexer TeD model, the user adds the Multiplexer image, with cell-loss performance statistics updated continually at runtime. On the client end, this is achieved by simply defining a sub-class of the TeD model view class (class CTedModelView), and overriding its cmd() and paint() methods. In the TeD model, an event is sent on the stdout channel whenever a cell is dropped by the Multiplexer.

Default Views and Controls

Even though it is desirable to provide the user with the ability to incorporate customized views, it is important to supply some default views, controls and other features that are common across most simulations and models. These features can be categorized into several domains, such as debugging, progress and performance monitoring, visualization of model semantics and points of interesting behavior, and so on. Determining the types of views that are most useful for each category is a rich research area, especially in the context of parallel simulation.

While supporting a set of simple views, Jane provides the architecture to plug-in more sophisticated views that are determined by other researchers. Among the current default simulation views and controls are:

- Simulation control (start, stop, pause, resume, kill)
- Simulation progress (simulation time advances, percentage of completion)
- Simulation performance (continuously updated aggregate and per-processor event plots, categorized as processed, committed, rolled-back and aborted)
- Simulation critical path analysis.

Default TeD specific views are also supported for simulations of TeD models. These include the display of the hierarchy of TeD entities, along with entity information such as their types and list of state variables. Other TeD specific displays include visualization of entity channel mappings, animation of event flows on channels, and animation of TeD process execution.

Synchronization

The interface between the parallel simulation and the interactive system must address synchronization issues, such as the method of controlling time advances, and the exchange of instrumentation data.

In Jane, the clients are implemented as conservative processes that are guaranteed not to incur any rollbacks. Instrumentation events from GTW/TeD simulations are sent to Jane clients using conservative I/O mechanisms of GTW. Event data is transferred from the simulation to Jane clients only when Global Virtual Time (GVT) reaches the receive-timestamp of the event.

Security

Security is an important consideration in the context of remote-access to simulations, which can make the remote computer susceptible to various kinds of security threats. Although Jane does not support total security in remote access, it implements a small set of measures to guard against the simple security problems. For example, the Jane server imposes a limit on the number or client sessions that can be active at any given time (currently a limit of 5 active clients is enforced). If a client tries to establish connection to the server at a

time when the limit is reached, the connection request is denied. Another simple security measure is to enforce a hard limit on the maximum elapsed/CPU time utilized by any simulation run. To prevent intentional or unintentional execution of executables other than bonafide simulations, the simulation executables can be restricted to be chosen from a suitably isolated directory. In addition, connections can be declined if they originate from machines that do not belong to a user-defined list of authorized machines. A consoling factor in the context of TeD simulations is that the security problems are lessened due to the fact that the TeD language currently does not provide any facilities to introduce executable content (such as macros) into the simulation beyond compile-time.

We are currently unable to use applets for implementing the clients (Jane clients are stand-alone Java programs) due to the limitation that applets can connect only to hosts from which the applet is loaded. Specific workarounds indeed exist, but we are investigating ways to deal with this problem in general.

Collaboration

Collaboration in simulation implies that different users cooperate in the execution of the same simulation. Geographically distributed users can benefit from the capability for collaboration, through which they all can observe identical displays of the same simulation, and control that simulation remotely to cooperate in debugging or monitoring. An important advantage of collaboration is that it facilitates having simulation experts help modelers debug their models or improve simulation performance. This is especially important in the context of parallel simulation because most modelers are not knowledgeable about the intricacies of parallel simulation, and hence the intervention of simulation experts is often needed to debug or fine-tune the model. Since users can connect to servers from remote locations, it is possible for the experts to join an on-going simulation session to cooperate with the users in detecting and fixing problems in the simulation model.

Collaborative systems, in general, adopt fixed protocols or metaphors for coordinating the actions of the collaborators in order to avoid inconsistencies and conflicts. Due to the nature of collaborative work in the context of parallel simulation, there is no clear distinction among the users to warrant a master-slave relationship. Moreover, users can join and leave at arbitrary points in time before, during or after the simulation. Hence, we chose to provide mechanisms for the users to communicate with each other, and leave coordination policies to the users. Since this can potentially lead to conflicts and inconsistencies in the simulation, we implemented almost all simulation actions as idempotent, i.e., multiple invocations of the same action are resolved such that only one of the invocations is executed, and the others become null operations. For example, when more than one user attempts to pause the simulation, all the pause commands are effectively reduced to one pause command.

Multiple Scenario Analysis

When simulation is used as a tool in decision-making (for example, in aggregate-level battlefield simulation), it is necessary to evaluate multiple scenarios simultaneously following a decision tree. In such situations, users interactively select branching points and perform what-if analysis along different decision paths. Implementation techniques such as cloning (Hybinette 1997) are used to efficiently support such interactive decision operations, significantly reducing the resource requirements as compared to running several independent and distinct simulations to evaluate multiple scenarios.

In the Jane environment, the clients, equipped with the interactive decision support interface, map the user actions into appropriate operations on the servers and/or multiple simulations that cooperate in the multiple scenario analysis. The Jane architecture acts as the glue in realizing this capability. We are currently working on enhancing the Jane clients to support scenario analysis using a combination of its scripting and graphical interface features, while incorporating the cloning technology into the parallel simulator.

Critical Path Analysis

Critical path analysis is a method of estimating the amount of parallelism present in a simulation, which gives a non-trivial upper bound on the speedup that can be expected for a parallel simulation model for a specified mapping to a set of processors. This can be a useful initial step in the process of tuning the application for better parallel performance. We have incorporated the critical path analysis algorithm of Lin (Lin 1992) into GTW, which computes an estimate of the ideal speedup achievable on any given GTW application. This feature has been useful in confirming that some of our network model simulations indeed achieve close to their estimated ideal speedup.

5. STATUS AND FUTURE WORK

Robust implementations of the server and Java client are currently operational, and network modelers are actively using the Jane environment for simulating and visualizing their models. The server is tested on Sun workstations and Intel PCs running Solaris, and SGI workstations and multi-processors running Irix. Java's portability permits the clients to be run on many machine platforms. Multiple geographically distributed users can run any GTW/TeD or RTI-Kit simulations in interactive and collaborative mode. The exact same copy of the server and client software can be used for both GTW/TeD as well as RTI-Kit simulations,

demonstrating the framework's simulator-neutrality. The server-simulation communication protocol is being documented to help developers of other simulators to interface with the Jane framework. Security and speed enhancements are being made, along with support for multiple scenario analysis through cloning.

REFERENCES

"The High Level Architecture", Defense Modeling and Simulation Organization, http://www.dmso.gov/hla.

Fujimoto, R., Ferenci, S., "RTI Performance on Shared Memory and Message Passing Architectures," 1999 Spring Simulation Interoperability Workshop, March 1999.

Scalable Self-Organizing Simulations Project (S3), http://dimacs.rutgers.edu/Projects/Simulations/darpa/.

On-line information on TeD and Jane, http://www.cc.gatech.edu/computing/pads/ted..html.

Lin, Y. 1992. "Parallel Analyzers for Parallel Discrete Event Simulation." *ACM Transactions on Modeling and Computer Simulation* 2, no. 3 (July).

Carothers, C., *et al*. 1997. "Visualizing Parallel Simulations in Network Computing Environments: A Case Study." In *Proceedings of the 1997 Winter Simulation Conference* (Dec).

Graham, J., *et al*. 1996. "A Visual Environment for Distributed Simulation Systems." *Simulation Digest*.

Graham, J., *et al*. 1998. "Evaluation of a Prototype Visualization for Distributed Simulations." In *Proceedings of the 1998 Winter Simulation Conference*.

The Maisie Visual Programming Environment, http://may.cs.ucla.edu/projects/mvpe/.

Franks, S. *et al*. 1997. "State Saving for Interactive Optimistic Simulation." In *Proceedings of the 11th Workshop on Parallel and Distributed Simulation* (June).

Hybinette, M. and Fujimoto, R. 1997. "Dynamic Virtual Logical Processes." In *Proceedings of the 12th Workshop on Parallel and Distributed Simulation* (June).

BIOGRAPHIES

Kalyan Perumalla is a Research Scientist and Ph.D. candidate at the College of Computing, Georgia Tech. His research interests include parallel telecommunication network simulation and parallel combinatorial optimization.

Richard Fujimoto is a Professor at the College of Computing, Georgia Tech. His research interests include computer architecture, parallel processing and parallel and distributed simulation.