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, web-based simulation, collaboration, 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. 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 the 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 to support the various phases of the modeling and simulation process 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 computing resources. 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[1]). 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[13]). 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 should be recognized 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 necessary for interactive systems to provide facilities for users to collaborate during model development and simulation execution. 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.

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[4]. 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)[6], 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[17]. The TeD modeling language[11][14] 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[3], 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[1]). Animations of processor behavior and performance visualization are presented in [2]. Performance issues related to interactive parallel simulation have been well-studied [2][5][7]. An empirical evaluation of the effectiveness of performance visualization features is presented in [8]. The state saving and performance issues have been studied in [5] 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[17]), and testing and improving them continually based on feedback from the users. Other recent efforts, such as the PowerSim Distribution Kit suite of commercial products for web-based simulation[16], are closely related to our work, although they seem to differ in the goals and mechanisms.

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. In section 5, we present case studies in which the use of Jane in actual simulations is discussed. 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, 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, permitting the independent operation of client, server, and simulator processes. Figure (i) 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.

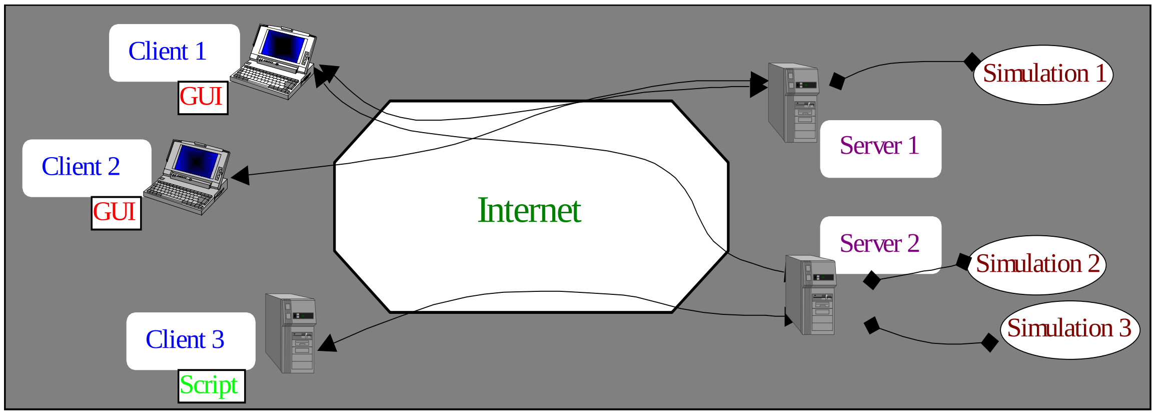


Figure (i): Client—Server Architecture for a simulation session in Jane.

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 at least one of the simulation processes be executed on the same machine as their 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 progress and 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.

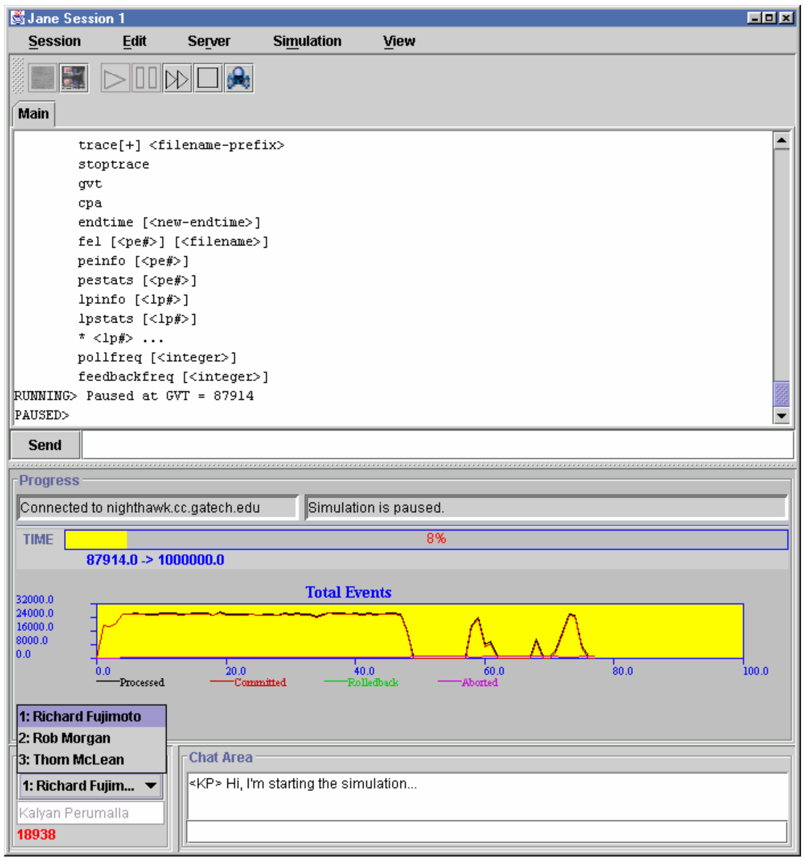
The client supports TeD-specific and RTI-Kit specific default views, which are automatically generated for any TeD model or RTI-Kit federation.

The Jane software includes a client that can be run as a stand-alone Java program. The client can also be run as an applet in a web browser, provided that the host machine of the Jane server has an active HTTP (Hypertext Transfer Protocol) server. The client supports a graphical user interface and includes a scripting API. Graphical interface is built using the Java Foundation Classes. The client in fact is a library of Java packages, each of which defines an API for a set of interaction operations. The Jane server and client communicate 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 (ii).

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.



```
        trace[+] <filename-prefix>
        stoptrace
        gvt
        cpa
        endtime [<new-endtime>]
        fel [<pe#>] [<filename>]
        peinfo [<pe#>]
        pestats [<pe#>]
        lpinfo [<lp#>]
        lpstats [<lp#>]
        * <lp#> ...
        pollfreq [<integer>]
        feedbackfreq [<integer>]
RUNNING> Paused at GVT = 87914
PAUSED>
```

Connected to nighthawk.cc.gatech.edu

Simulation is paused.

87914.0 -> 1000000.0

1: Richard Fujimoto
2: Rob Morgan
3: Thom McLean

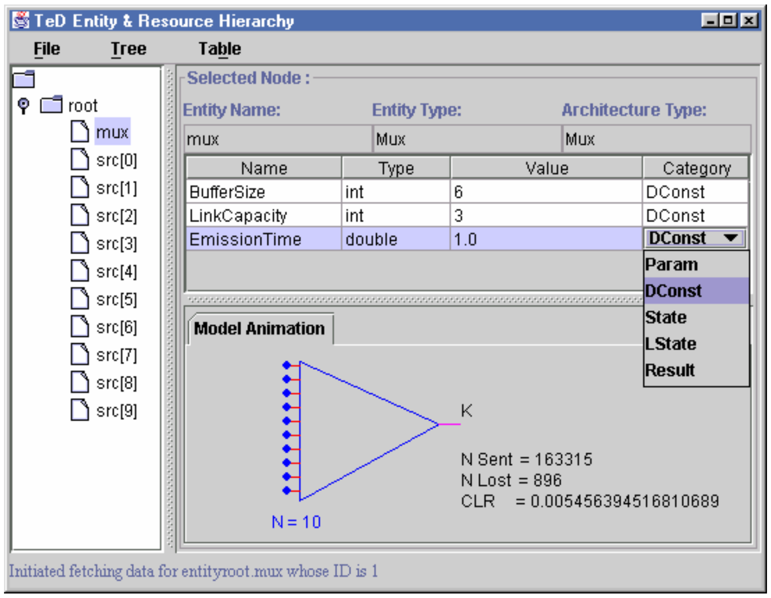
1: Richard Fujim...

Kalyan Perumalla

18938

<KP> Hi, I'm starting the simulation...

Figure (ii): Main Jane client window showing a sample collaborative session



Entity Name:	Entity Type:	Architecture Type:
mux	Mux	Mux

Name	Type	Value	Category
BufferSize	int	6	DConst
LinkCapacity	int	3	DConst
EmissionTime	double	1.0	DConst

N Sent = 163315
N Lost = 896
CLR = 0.005456394516810689

Initiated fetching data for entityroot.mux whose ID is 1

Figure (iii): 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 (iii)). 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 (iii) 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 that are 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 Java 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 (iii), for example, the entity tree and the table of variables are 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 pre-defined 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 TeD entity hierarchy, along with entity information such as the types and values 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.

When a client receives an instrumentation event, application-specific processing can be performed, such as scheduling animation actions based on that event. To facilitate in displaying smooth animation actions according to logical time order, a time-stepped animation package has been developed and included with the client software. This package supports scaled real-time animation of actions such as object motion and scaling, and at the same time stays synchronized with the simulation. The synchronization of the animation with the simulation is achieved by animating only those actions whose time stamps are less than the current GVT value.

The clients can also perform read and write operations on the simulation state variables. However, due to the multi-threaded nature of parallel simulation, the simulation needs to be paused before any write operations, such as changing the value of a state variable, are performed. The framework is also capable of inserting new simulation events generated by the clients into the simulation at runtime, although such capability is not yet implemented in the current version. In the case of optimistic simulation, inserting new events is relatively straightforward as long as the effect of the new event on the GVT computation is carefully taken into account.

Safety across Layers

Modern simulation software is organized as several layers, such as simulator kernel, simulation object library, domain-specific model repository and user-defined model configuration. For example, the TeD system consists of the following layers:

- The GTW parallel simulation kernel, supporting the logical process abstraction
- The C++ class library that is used to realize TeD entities as GTW logical processes
- The TeD model repository, such as ATM PNNI or wireless network models
- Specific network configurations which are instantiations of the TeD models.

Typically, the user requires graphical visualization of various activities of all the layers, with varying degrees of detail for each layer. When such visualization needs to be supported remotely (for example, over the Internet), it becomes necessary for these different layers to share the communication channel from the simulation to the visualization client. This is because, in practice, it is not feasible to provide an independent communication channel (such as a TCP socket) for each layer.

In such a situation, the integrity of data transferred across the communication channel must be protected in the presence of errors introduced by any of the layers that access the channel. For example, in the case of a stream-based channel (such as a TCP socket), if the data format of a message is not accurately matched at both the sending and receiving ends, then errors may result. If all the message data is not extracted, then the extra data erroneously spills over to the next message. Similarly, the data of the following message may erroneously be extracted as part of the current message. Data mismatch can occur either due to programming errors of omission and commission, or due to the heterogeneity of server, client and simulation platforms with respect to byte ordering and packing. Although such errors can be minimized with adequate care in the case of kernel and simulation object layers (on both the client and simulation sides), they cannot be discounted in the user-defined instrumentation of user models for model-specific visualization. It is important that the client, server and the simulation continue to function in the presence of such errors. In Jane, this safety issue is addressed using a *channel access interface* as described next.

In Jane, all modules that need access to the underlying client-server-simulation communication channel (socket or pipe) for receiving or sending data must do so with the help of a clearly defined interface library called the *record stream* library. Logically, the record stream library supports an ordered datagram service with unlimited packet length and with automatic conversion of data representation. In other words, this library supports two key features: (1) safely enforces record boundaries of messages, without limiting the record length (2) automatically converts the data representation between heterogeneous platforms. The library hides the internals of the communication channel, but provides a streamlined function call interface for operating on the channel. The interface supports operations such as preparing the channel for sending or receiving a record of data, querying or specifying the types of record data fields, followed by sending or receiving the data, and ending the sending or receiving operation. The library implements safety by inserting and detecting record boundary tokens in the channel, and implements data conversion by using ASCII as the intermediate format for actual data exchange over the channel. This ensures data integrity across simulation and modeling layers, as well as supports data transfer between the simulation written in C and the client written in Java despite differences in their internal data representations.

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.

Application vs. Applet

By default, the Jane client is executed as a stand-alone Java application. Alternatively, it is possible to execute the client as a Java applet inside a conventional web browser. Both approaches have their advantages and disadvantages, and hence the user must make the choice between the two depending on the user's specific requirements.

For stand-alone clients, the client software must be installed and customized to the user's environment. In practice, we have observed that the installation and customization procedure can be a hurdle for new users. Applets, on the other hand, require little user-intervention for installation, since the browser automatically downloads and runs the applet code. This can be a very appealing feature for uninitiated users. Applets, similarly, are excellent means for quickly deploying upgraded versions of the client software.

Applets, however, are only authorized by default to connect to the host from which the applet code has been downloaded. This implies that, in order for the client applets to connect to the Jane server, the applet code must be downloaded from the Jane server host. This in turn implies that the Jane server host must also run an HTTP server, which can potentially increase the processing load on that host. Alternative workarounds exist by using digital signatures, but are difficult to configure by typical users.

An important issue concerns the recording of user preferences, such as simulation parameter settings. It is necessary to save such settings across sessions, so that the user does not need to explicitly type in the same values for every session. Stand-alone Jane clients save such information in a profile file on the client host, which is read and updated during each session. It is our experience that this is a feature required by all users. Applets, on the other hand, are not typically permitted to read and/or write to local disk, implying that the profile file cannot be read or saved across sessions at the client host.

Another issue surrounding applets is the fact that the browsers do not always keep up with the latest versions of Java. For example, the Java Foundation Classes are not by default supported in all the browsers. Although this can be solved by customizing the user's browser as needed, it requires user intervention, which is unappealing. The Java plug-in methodology that is recently becoming popular can be of help in this regard.

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[9] 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[10] 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. CASE STUDIES

The Jane system is being used to provide interactivity for both parallel telecommunication network simulations as well as federated simulation applications such as battlefield simulation. Some representative scenarios of interactive simulation in such applications are described in this section.

Federated Battlefield Simulation

The RTI-Kit is being used as a research tool in developing techniques for high-performance implementations of the HLA RTI, using specialized computer interconnects such as the Myrinet switches[6]. A battle tank simulation has been developed as an application for benchmarking the performance of HLA federations based on the RTI-Kit. This simulation consists of several federates (autonomous simulation processes) each of which simulates one or more battle tanks. The tanks engage in attacks when they come within enemy range. All communication among federates happens through the HLA RTI. Each federate runs on a processor dedicated to that federate.

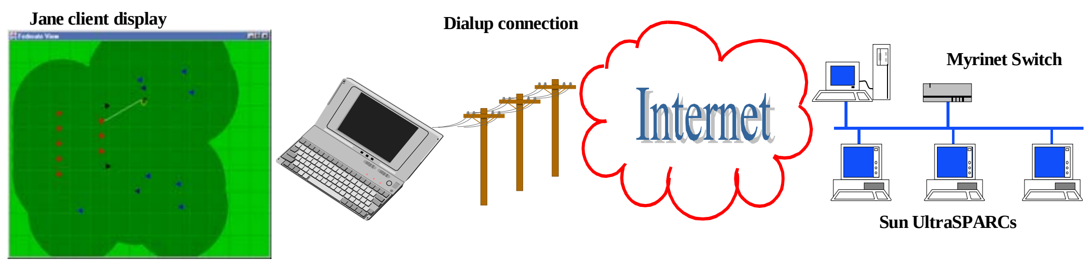


Figure (iv): Remote execution of a federated battlefield simulation using Jane

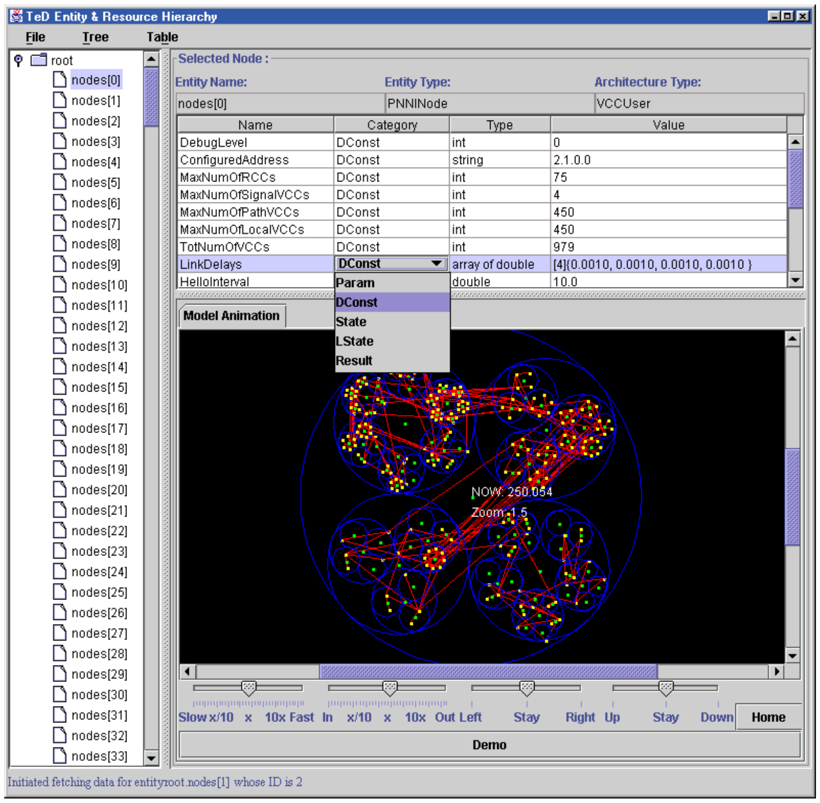
Two federates are responsible for simulating the two opposing sets of tanks, while another federate simulates their platoon leaders. A separate federate serves the purpose of listening to activities of interest, and communicating them to the Jane client. Events of interest include tank position updates, enemy detection, fire and target destruction. Graphical visualization of such events is realized as application-specific extension to the Jane client (see Figure (iv)).

Jane has been useful in several ways in the development and usage of this benchmark application. First, the graphical visualization helped in exposing certain bugs during initial application development. For example, noticeable abrupt jumps in the tank movement pointed to coding errors in the model. Jane has been useful for visually comparing and confirming the performance improvements attributable to novel techniques such as "approximate time order" scheduling incorporated into the RTI. For example, the improvements were made evident from observing the runtime plots of the rate of simulation time advances.

Jane was also helpful in interactively demonstrating this application to the project sponsors. The remote access feature of Jane has been useful during such demonstrations that were held in Washington D.C[11]. The Jane client was used to connect over a dial-up telephone line to the Internet, and the simulations were executed remotely on a Myrinet-connected cluster of Sun workstations at Georgia Tech in Atlanta. Although the simulations ran faster than real-time, the amount of network traffic from the simulation to the client was kept within the available bandwidth of the dial-up telephone connection by sampling the visualized events at a fixed rate (approximately 30Hz). This resulted in considerably less network traffic without sacrificing the perceptible fidelity of the visualization.

Parallel Network Simulation

The interactivity features of Jane are also being applied in our telecommunication network simulation projects, whose primary focus is on the efficient simulation of large-scale network configurations modeled using the TeD language. As an illustration, Figure (v) depicts the use of Jane for visualizing TeD models of ATM (Asynchronous Transfer Mode) inter-networks that conform to the PNNI (Private Network to Network Interface) protocols[14].



Selected Node :

Entity Name: nodes[0]
Entity Type: PNNINode
Architecture Type: VCCUser

Name	Category	Type	Value
DebugLevel	DConst	int	0
ConfiguredAddress	DConst	string	2.1.0.0
MaxNumOfRCCs	DConst	int	75
MaxNumOfSignalVCCs	DConst	int	4
MaxNumOfPathVCCs	DConst	int	450
MaxNumOfLocalVCCs	DConst	int	450
TotNumOfVCCs	DConst	int	979
LinkDelays	DConst	array of double	[4]{0.0010, 0.0010, 0.0010, 0.0010 }
HelloInterval		double	10.0

Initiated fetching data for entityroot.nodes[1] whose ID is 2

Figure (v): Client-side visualization of a parallel ATM PNNI network simulation

For all TeD models, Jane automatically generates a client-side display of the entity hierarchy tree, along with the configuration information, such as parameters and state variables associated with each instantiated entity. The client-side display of TeD entity hierarchy requires the transfer of the hierarchy specification and associated information from the simulation to the client after the network has been instantiated and initialized in the simulation. In practice, transferring such data for all entities during initialization consumes significant amount of time, especially in the case of networks containing large number of entities. To address the problem of the initialization time, an alternative approach is used in which the entity information is fetched on the fly, on an as-needed and when-needed basis. During initialization, only the root entity information is fetched automatically. For all other entities, the user can click on an entity icon to expand its sub-tree and fetch that entity's parameters and state variable information at runtime.

Since most of the TeD models are analytical in nature, real-time issues were not relevant on the client-side. However, in most models, it was important to carefully choose the set of instrumented simulation events that were to be sent to the client, to prevent exceeding the bandwidth of the network connection from the simulation to the client.

6. STATUS AND FUTURE WORK

Robust implementations of the server and Java client are currently operational, and the Jane environment is actively being used for simulating and visualizing parallel network simulations as well as federated simulation applications. The server is tested on Sun workstations running Solaris, Intel PCs running Solaris or Linux, and SGI workstations and multi-processors running Irix. The clients have been tested on Sun and SGI workstations, as well as desktop and laptop versions of Intel PCs and Apple Macintosh. Java's portability permits the clients to be run on several additional machine platforms. Multiple geographically distributed users can run any GTW/TeD or RTI-Kit simulations in interactive and collaborative modes. 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 further refined and 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. The facilities for collaborative simulation are being exercised on a wider range of simulation applications, and enhancements will be made based on the experiences gained.

ACKNOWLEDGEMENTS

This work has been partially funded by the NSF research infrastructure grant CDA-9501637. We would like to thank Rob Morgan for his help in developing parts of the Jane client graphical interface. We would also like to acknowledge the helpful feedback on initial versions of the Jane environment from our project partners.

REFERENCES

[1] Bagrodia, R., The Maisie Visual Programming Environment, Department of Computer Science, University of California, Los Angeles, http://may.cs.ucla.edu/projects/mvpe/.

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

[3] Das, S., *et al*. "GTW: A Time Warp System for Shared Memory Multiprocessors." In *Proceedings of Winter Simulation Conference* (December 1994).

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

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

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

[7] Graham, J., *et al*. "A Visual Environment for Distributed Simulation Systems." *Simulation Digest (1996)*.

[8] Graham, J., *et al*. "Evaluation of a Prototype Visualization for Distributed Simulations." In *Proceedings of the 1998 Winter Simulation Conference* (December 1998).

[9] Hybinette, M. and Fujimoto, R. "Dynamic Virtual Logical Processes." In *Proceedings of the 12th Workshop on Parallel and Distributed Simulation* (June 1998).

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

[11] Loper, M., "Exploiting Temporal Uncertainty to Relax Ordering Constraints," DARPA Advanced Simulation Technology Thurst Project Brief, Arlington Virginia (April 1999).

[12] Nicol, D., Editor, "Special Issue on TeD," *SIGMETRICS Performance Evaluation and Review*, 25(4) (March 1998).

[13] Panchal, J., *et al.* "WiPPET, A Virtual Testbed for Parallel Simulations of Wireless Networks." In *Proceedings of the 12th Workshop on Parallel and Distributed Simulation* (June 1998).

[14] Perumalla, K., Andrews, M. and Bhatt, S. "TeD Models for ATM Internetworks," *SIGMETRICS Performance Evaluation and Review*, 25(4) (March 1998).

[15] Perumalla, K., Fujimoto, R., "On-line information on GTW, TeD and Jane," College of Computing, Georgia Institute of Technology, http://www.cc.gatech.edu/computing/pads/ted..html.

[16] The *PowerSim* Business Simulation Products, http://www.powersim.com.

[17] The Scalable Self-Organizing Simulations Project (S3), Wireless Networks Laboratory and DIMACS, Rutgers University, http://dimacs.rutgers.edu/Projects/Simulations/darpa/.

BIOGRAPHIES

Kalyan Perumalla is a Research Scientist since 1997 at the College of Computing, Georgia Institute of Technology, where he is also finishing his doctoral degree in Computer Science. He received the B.E. degree in Mechanical Engineering in 1991 from Osmania University, India, and the M.S. degree in Computer Science in 1993 from the University of Central Florida, Orlando. Previously, he worked at the Institute for Simulation and Training, Orlando in 1992-93, and interned at Schlumberger in 1994 and at Bellcore in 1995 and 1996. He received the best paper award at the 1999 Parallel and Distributed Simulation Workshop (PADS'99) as co-author of a paper on efficient parallel simulation. His current research interests include parallel and distributed simulation, telecommunication network modeling and simulation, and parallel combinatorial optimization.

Richard Fujimoto is a professor in the College of Computing at the Georgia Institute of Technology. He received the Ph.D. and M.S. degrees from the University of California (Berkeley) in 1980 and 1983 (Computer Science and Electrical Engineering) and B.S. degrees from the University of Illinois (Urbana) in 1977 and 1978 (Computer Science and Computer Engineering). He has been an active researcher in the parallel and distributed simulation community since 1985 and has published over 100 conference and journal papers on this subject. He has given several tutorials on parallel and distributed simulation at leading conferences. He has co-authored a book on parallel processing and recently completed a second on parallel and distributed simulation. He served as the technical lead in defining the time management services for the DoD High Level Architecture (HLA). Fujimoto is an area editor for ACM Transactions on Modeling and Computer Simulation. He also served as chair of the steering committee for the Workshop on Parallel and Distributed Simulation, (PADS) from 1990 to 1998 as well as the conference committee for the Simulation Interoperability workshop (1996-97).