Proceedings of the 2000 Winter Simulation Conference
J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, eds.

OBJECT-ORIENTED SIMULATION OF DISTRIBUTED SYSTEMS USING JAVA® AND SILK®

Richard A. Kilgore

ThreadTec, Inc.
P.O. Box 7
St. Louis, MO 63006, U.S.A.

Emmett Burke

Symbi Systems, Inc.
816 SW Normandy Terrace
Seattle, WA 98166, U.S.A.

ABSTRACT

An object-oriented modeling infrastructure using the Java-based, Silk simulation classes is defined that enables the simulation of multitasking, distributed systems using symmetric multiprocessors. The simulation infrastructure is being used to evaluate alternative architectures for embedded, distributed systems. We show how the underlying structure is adapted to several different applications, including various Internet applications. The paper describes the infrastructure, its robustness, and the application of the model to produce insights for a system under design. The simulation infrastructure enables a high fidelity representation of the internal complexity of the application on each processing node, the operating system behavior, and the disks and network. The simple yet powerful representation leverages the use of the Silk entity-thread architecture to achieve a simulation architecture that maps to the actual system architecture in both conceptual design and processing sequence. The model has been validated through instrumentation of the evolving target system.

1 BACKGROUND

Current information systems architectures are generally comprised of multiple computational nodes cooperating in a predefined manner. Distributed systems have been quite common in the defense world, evolving to enterprise architectures, and are now seen commonly in Internet applications as well as many consumer items like automobiles. It is critical to get the architecture of these systems correct early so as to avoid costly rework and lost schedule. The architecture of these systems is much more than the topology of the nodes or the power of the nodes. While the topology is important, equally important is the design of the processes and the threads that reside in them, the characterization of the thread behavior, their scheduling algorithms, and defining the inter-process communication mechanism. Simulation tools enables the architects to assess radically different designs, and choose the one that has the best temporal behavior for the application. With a selected architecture with known latency characteristics, one has a basis for the creation of performance budgets for components of the system and thus is in a better position to manage the development to successful completion. The focus of many simulations has unfortunately been on the narrower issues of distributed systems like network performance (George et al, 1999). The system that is described here encompasses the network topology, but includes the thread architecture of each node and the critical inter-process communication structure. It is built on a set of reusable components situated in a software development environment that supports component reuse (Pidd, 1999).

The specific distributed system described in this application is composed of a central node consisting of a shared memory multiple processor computer. Other nodes, that also consist of single and multiple processor computers, are connected via a high bandwidth local area network. The system uses a threaded software design with Orb-based inter-process communication between lightweight processes. A SCSI disk backup of data and transactions processed by the system is an important part of the system design as these backup processes are processor intensive and network intensive and are large contributors to message latency. The system design requires scheduling of software processes and design of message buffers. The timing of these scheduling decisions and the sizing of these message buffers determine the ability to achieve rigorous performance targets for resource utilization and message latency.

In Section 2 of this paper, we discuss the requirements of the model-based decision support for distributed systems and the objectives of object-oriented simulation support. Section 3 is an overview the design of the simulation objects and the mapping of the Silk classes to these simulation objects. Section 4 describes features of Java and Silk that enables high fidelity visualization of distributed system simulation. Section 5 concludes the paper with an example of the statistical results and some discussion about the benefits and challenges of this approach to distributed system simulation.

2 OBJECT-ORIENTED MODELING APPROACH

An important result of this work was the achievement of a true implementation of object-oriented simulation design and execution using Java and the Silk classes (ThreadTec, 1997). The Silk classes are a Java Application Programming Interface (API) for general-purpose, discrete-event, process-oriented, object-oriented simulation (Healy and Kilgore, 1997). It was felt that the real success of the modeling activity would be dependent upon two features of the Silk/Java and the object-oriented simulation approach (Joines and Roberts, 1996.). The first was the ability to quickly adapt the behavior of the object-oriented modeling components to ever-changing assumptions and architectural changes. The second was the ability to accurately represent the system behavior at the right level of fidelity of the system components and accurately collect statistical information of the impact of system design on message latency. It is necessary to understand each of these simulation design requirements to better appreciate the Silk implementation described in later sections.

Simulation in support of distributed system design requires the ability to change the model frequently in response to new ideas and suggestions. The process of distributed system design usually involves the coordination of several vertical and horizontal layers of expertise involving different company divisions and even different companies. The required credibility and the timeliness of the decision support for this function creates challenges for the simulation developer who must anticipate the need for intermediate decisions with imperfect information. It also requires anticipation of the continuous refinement of system component definitions and behavior. The ability to meet these modeling challenges rests in the ability to decompose the distributed system into independent modeling components that allow a component-based view of the system. These modeling components must contain interfaces that are independent of the structure of downstream and upstream components.

As the distributed system hardware design matures, subtle decisions regarding software design begin affecting hardware performance and message latency begins impacting areas that intuition would not expect to be impacted. It is critical that the modeling components designed can accommodate increasing levels of fidelity regarding the definition and behavior of the corresponding system components. It is equally critical that precise statistical information regarding system performance, and the reasons for the performance, be generated by the simulation.

In this case, the ability to program in Java and Silk was essential for creating the level of fidelity necessary to properly define the nature of system delays and the impact of software changes on these delays. Other simulation software approaches may have offered easier initial construction of the simulation skeleton. But none were able to flesh-out the skeleton without restrictions than that possible in a full-featured object-oriented programming language. Finally, the ability to animate the identity of each system message, process and system component in detail was essential in providing explanations for the unexpected delays that were observed in the summary statistical information. Without this ability to simultaneously view multiple systems components and individual messages dynamically, the conclusions made from the statistical summaries were often erroneous.

3 THE MODEL

In the process of modeling the incremental development of a complex command and control system, we have developed a set of classes that enable us to model messages, threads, light weight processes, and symmetric processors. We have built on the Silk infrastructure of entity-threads, animation, statistical reporting, and event scheduling classes to produce appropriate statistics that characterize the temporal behavior of the system.

From the point of view of a distributed systems application, one typically defines a set of processing nodes, which are often symmetric processors, and the network topology that connects the nodes. Then, within each processing node, one defines the processes, and within a process, the threads of that process. The lightweight processes are operating system facilities to which threads are mapped. Lightweight processes are the units that the operating system schedules for execution on the processor. Messages are created either externally or internally. Messages may enter the system from external hardware or may be created in the activity of a thread processing another message. Messages are routed to other threads by being placed on their input queue. A message is defined by the threads or hardware, such as disks, that it visits. Latency is defined as the time for a message to visit a defined set of connected threads and resources.

The mapping of these system components to Silk classes is partially described in Figure 1. The actual definition of the target distributed system is beyond the scope of this paper but the principal results can be presented using this level of decomposition. The remainder of this section describes the necessary function of each class and the simulation requirements of the Silk implementation.

3.1 Message Class

The Message class is the foundation class for representing the data passed between processes and systems. The Message class models the queueing of a message onto the input queue of a thread. Although large varieties of message types exist in distributed systems, there are many common characteristics and behaviors. A common Silk Message class was created as a parent class which allowed

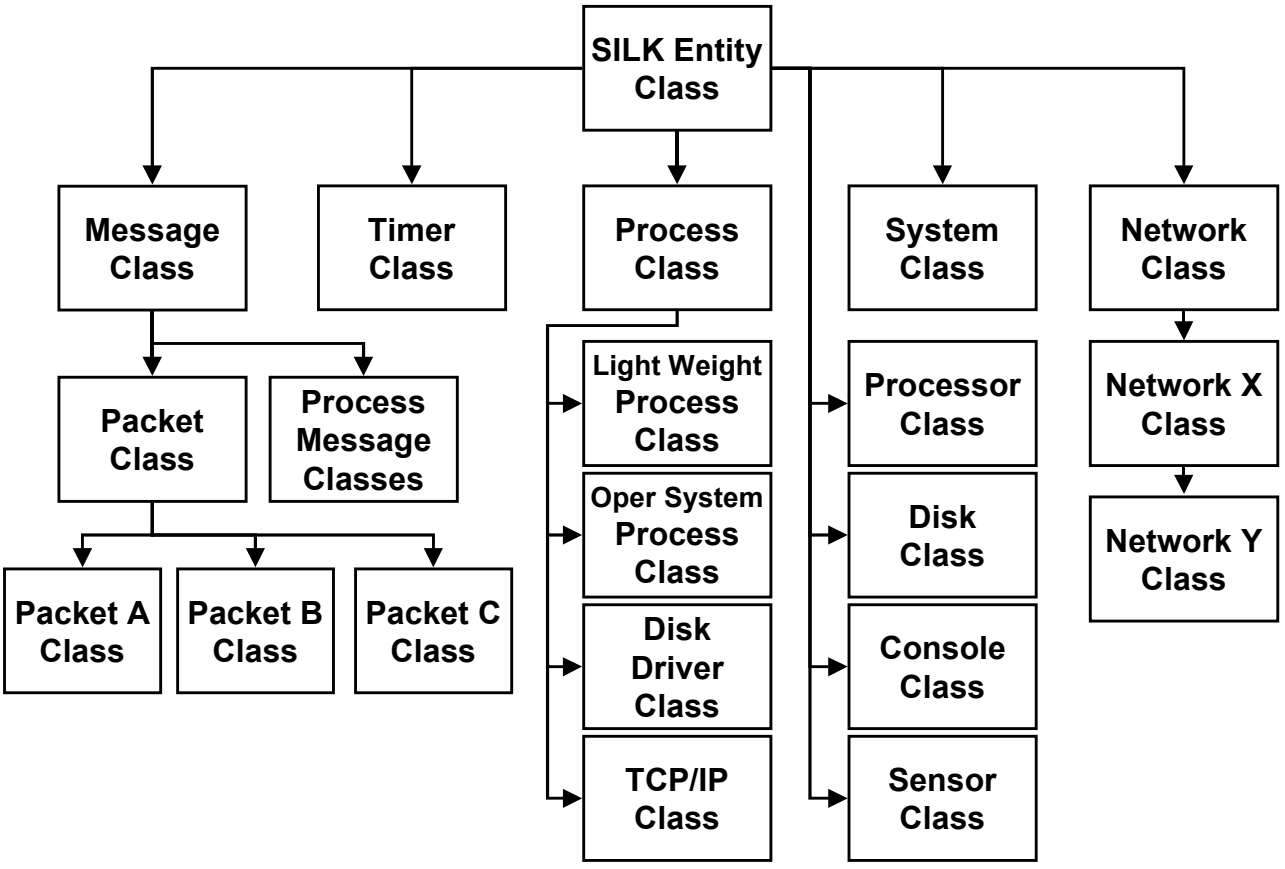

Figure 1: Silk Class Hierarchy for Distributed Systems Simulation

specific subclasses of data, network and operating system messages to be created without the re-definition of common features. The Silk language allows for extension of the basic Silk process-oriented methods common to discrete-event languages (queue, seize, delay, release, etc.). The Message class extended the Silk Entity methods to create a set of message-specific behaviors (generate, execute, send, updatestats) that greatly simplified the representation of the message sojourn through the system. As shown in the simplified code example in Figure 2, the Silk language supports a readable structure for describing message behavior uncommon in object-oriented simulation. A particularly useful capability was the dynamic creation of new simulation entity objects throughout the simulation as opposed to the typical requirement of creating arrays or collections of available objects prior to the simulation.

3.2 Process Class

The execute method of the Message class in Figure 1 refers to the execution of this Message within a subclass of the Process class. The lightweight process (Lwp) class models the acquisition of a processor, the processing of messages and the reschedule logic of the thread. An Lwp class incorporates the functionality of both the thread and the lightweight process. It acquires the processor by placing itself on the queue of the processors. Once it has acquired a processor, it performs the logic of the thread. This logic may be simple, as in the case of a pure input thread that may acquire the data from an object request broker. The time delay associated with the thread executing the message on the processor may be represented in a variety of ways.

```
public class Message extends Entity{
    double attStartTime;

 public void process{
 }

    public void execute( Message msg, Lwp lwp){
        msg.queue( msg.lwp.queue);
        msg.halt( );
        msg.seize( msg.lwp.resource );
        msg.delay( lwp.lookup( msg.className );
        msg.release(msg.lwp.resource )
        msg.updatestats( );
    }
}
public class Packet extends Message{
    public void process{
        attStartTime = time;
        execute( lwpa01 );
        execute( lwpa02 );
        spawn( "PacketA" );
    }
```

Figure 2: Extending Silk Objects and Methods

The parent of the Lwp class is the Process class that describes common characteristics for a large variety of software processes that execute on distributed, multi-processor systems. The utilization of the processor is dependent on the amount of processing required for the message served by these processes and the method in which the process itself is implemented within the operating system/processor combination. While the internal function of these software processes are unique, many share the same basic functions of accepting messages into process buffers, activating themselves for a period of time based on system clock (the Timer class) delays, processing message information, and then sleeping for a scheduled time period until the prioritized process is reinserted into the Processor queue for execution. Consequently, a parent Process class provides all of the foundation data characteristics and behavior methods common to the various Lwp, Operating System, network and disk driver subclasses required.

A consistent object-oriented design pattern between Messages and Processes is central to the ability to quickly modify the model as new logic is added to the process software or new message types are added to the system design. In Java, all of the characteristics and behavior are encapsulated within a single class and a single file that allows for a modular Silk model structure. The interface between these Silk classes is accomplished through shared queues which are identified only by the class name characteristic. Thus, when an additional Lwp class is created, the class name string is the only information necessary for all of the Messages that use this Lwp. All of the delays, queues and resources which the Messages use within the Lwp are generic references from this class name string which is used to lookup and return the Lwp object.

Likewise, the simulation description of the Lwp object is independent of the types of Messages that it is executing. The Lwp simply pulls the next Message from the interface queue, processes it, and then activates the halted message when complete.

This ability to freely mix "entity-push" and "resource-pull" behavior between classes and within classes in Silk-based models is ideal for valid representation of distributed systems logic. Traditional process-oriented simulation is typically restricted to an "entity-push", assembly-line world view where the focus is on transient entities whose aim is to complete a series of process steps. Also typical of traditional process-oriented simulation is the representation of passive, unintelligent, capacity-limited resources. For simplistic models, this approach is often sufficient, but in multitasking, distributed architectures, there is usually a need to model active, intelligent resources that take control of decision making within the system.

3.3 Processor Class

This importance of this transfer of control is even more evident in the interaction between the Process class and the Processor class. The Processor class represents the symmetric processors. Each of the operating system and lightweight processes must register their interest in being processed by insertion into the Processor queue. Depending on the number of processors, one or more idle entities continuously check this queue for idle processes that desire to become active. As shown in the simplified code example in Figure 3, the Silk entity-thread representing the processor continuously repeats this relatively simple process. It begins by using the Java/Silk while (condition( )) construct to stop the execution of the Silk entity-thread until the condition becomes false indicating that an Lwp has now joined the common Processor queue. The Processor then removes the Lwp from the queue and obtains a reference to the Lwp object. After changing the status of the processor for statistics collection, an initial delay for context switching is performed by the Processor.

```
public class Processor extends Entity{

    Lwp entLwp; //reference an Lwp object
    public void process{
    while (true) {
     // wait while processor queue is empty
     while(condition(queProcessor.getLength()==0));
     entLWP = ( Lwp )queProcessor.remove( 1 );
     seize( resProcessor );
     delay( varContextSwitchTime );
     entLWP.attProcessor = this;

        // activate lwp to simulate appropriate delay
        entLWP.activate( );
        // halt processor until lwp complete
        halt( );
        release( resProcessor );
    }

}
```

Figure 3: Silk Processor Logic Example

It is at this point that the transfer of control returns to the Lwp as the processor delay is actually dependent on what the Lwp process and message requirements dictate. The Processor entity is halted until the Lwp delays are complete. Note that a reference to the Processor instance assigned to this Lwp is stored as a property of the Lwp object before the Lwp object is activated. This allows the Lwp to activate the correct Processor object once the processing delay is complete.

Processor utilization is modeled as a function of the message type, its contents, and the thread that is executing the message. This provides exceptional versatility and is tied explicitly to validation of the target system through instrumentation. Also incorporated into the Lwp class is the ability of this class to give up the processor on a regular basis so that preemption by other threads of higher priority

may take place. An Event Trace Diagram using the Unified Modeling Language (UML) (Rational Software, 2000) is presented in Figure 4 that summarizes the interchange between the Silk classes.

4 DISTRIBUTED SYSTEMS VISUALIZATION

As mentioned earlier, an important function of distributed systems simulation is the presentation of the interaction of the system components to component experts. Each of these experts understands the operation of a specific component, but must rely on verbal or written descriptions of the interactions between components. For this reason, the visualization of the executing Silk simulation of the distributed system becomes a necessary part of the verification of the model. Silk employs JavaBean components to support the animation of executing Silk models through the extension of Java graphical objects such as text areas, progress bars and other multimedia APIs. This section describes the approach used to communicate the internal details of the model to the system engineers via animation.

The screen image in Figure 5 is typical of the overview animation screen that served as the portal for navigating through the component and message animations. Using common Java button objects as labels, this overview screen allowed the viewer to click within any particular component of interest to bring up a more detailed representation of the current state of the component. The ability to follow individual messages through the system and understand the impact of message delays elsewhere in the system requires this ability to zoom in and out of processes frequently.

A particularly innovative feature of distributed systems animation enabled by Silk and Java is the dynamic creation of Message entity icons that could be uniquely labeled or colored based upon Message class, source creation time, or latency. Rather than static images, these icons could be dynamically changed throughout their sojourn based on a common symbol attribute that was updated based on the individual characteristics of each Message entity. This ability to visualize messages with dynamic status information was an enormous aid in verification of the model by the simulation developers. It also allowed the investigation of non-intuitive statistical results that occurred since it allowed for the tracking of individual message latencies and the system state which caused these latencies to occur. This provided direct feedback to the system architects and software developers and provided a test bed for timely and economical analysis of the alternatives proposed to improve performance.

In addition to animation, detailed traces were created to log the interaction of system components both on the screen and to log files. In addition to the overall log file that summarized all system activity at the macro level, individual process and message logs were created to allow

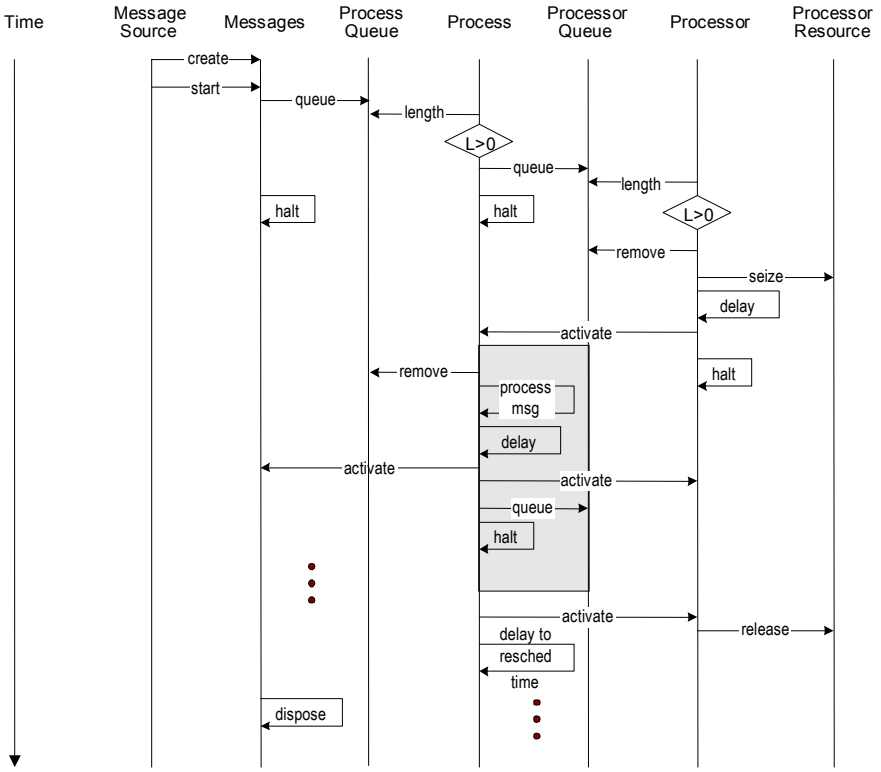

Figure 4: UML Event Trace Diagram of the Interaction of Silk Classes for Distributed Systems

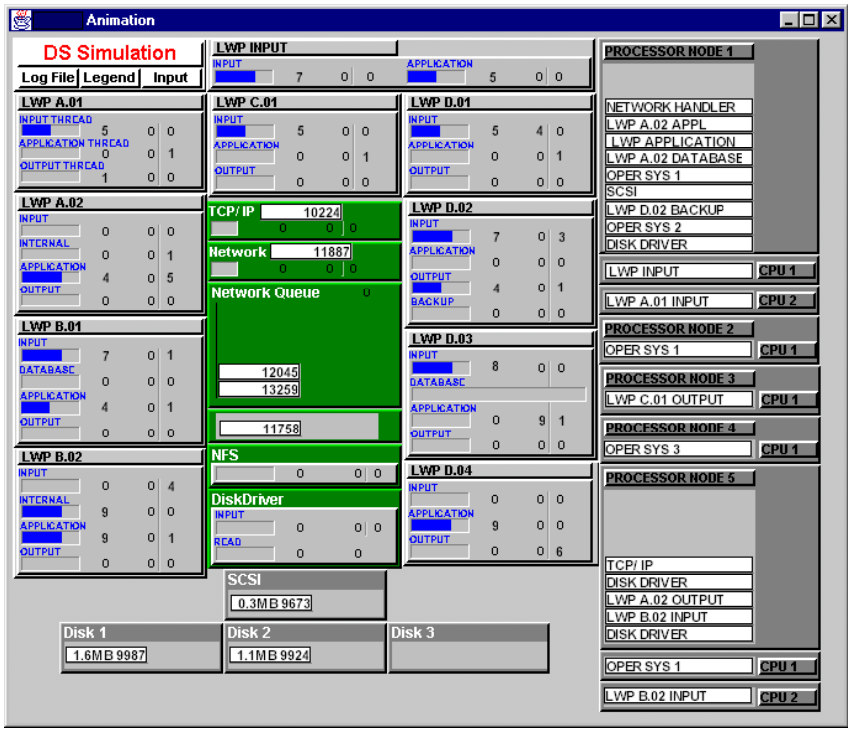

Figure 5: Animation of Silk Simulation of Distributed System

convenient tracking of behavior by component type. An important feature of these logs was that all entities are uniquely identified using the same identifiers as used in the executing Java/Silk program. Thus, model verification and debugging could be accomplished directly within the base language using professional-quality debugging tools, rather than in the tools and trace provided by a simulation language and language-specific simulation environments. This removed of an entire layer of model verification errors (from simulation language to underlying programming language). And the use of an industry-standard programming language and programming interface allowed for "non-simulation-trained" programmers involved in system software development to participate in the simulation verification and debugging activity. It is much more likely that these same engineers can be used for model maintenance and updates in the future than if a special-purpose simulation language or simulator was employed.

The present state of Java performance is vastly improved since it origins in 1995, but animation of this complexity required a minimum 256MB of RAM and 600+ MHz processors for simulations with hundreds of concurrent messages in the system.

5 RESULTS

There are two significant results of the effort to date. For the target distributed system, a great deal of useful information is being generated by the model on the desired performance measures of processor utilization and message latency. The Excel chart shown in Figure 6 is typical of the simulation output for message latency which identifies the queueing and processing delays by message throughout the simulation. It is important to note in the chart that each component of the system sojourn time represents the prototype instrumentation results. A particular difficult statistic in these types of models is the proper allocation of latency to the actual source of queueing delays since messages wait for both processes to be scheduled and processors to be available. The ability of Silk entities to exist in an unlimited number of queues simultaneously was helpful in the collection of this information.

The second result is that the investment in the generic structure of the model has paid off in that it has been easily adapted to the simulation of other message-based systems such as the modeling of web-server software and database processing involved in internet transaction processing (Banga, 1997; Burke and Kilgore, 2000; Hu et al, 1998). While the ability to reuse Silk simulation objects for these extensions is still not without code modification, the use of consistent object-oriented design patterns has made this task much simpler and straightforward than previous efforts with procedural-language-based, process-oriented languages. It is expected that the growth of interest in modeling and simulation of distributed architectures for web-based transaction processing will continue. The application of technologies and tools from simulation of signal processing and communication networks is an obvious foundation for this work.

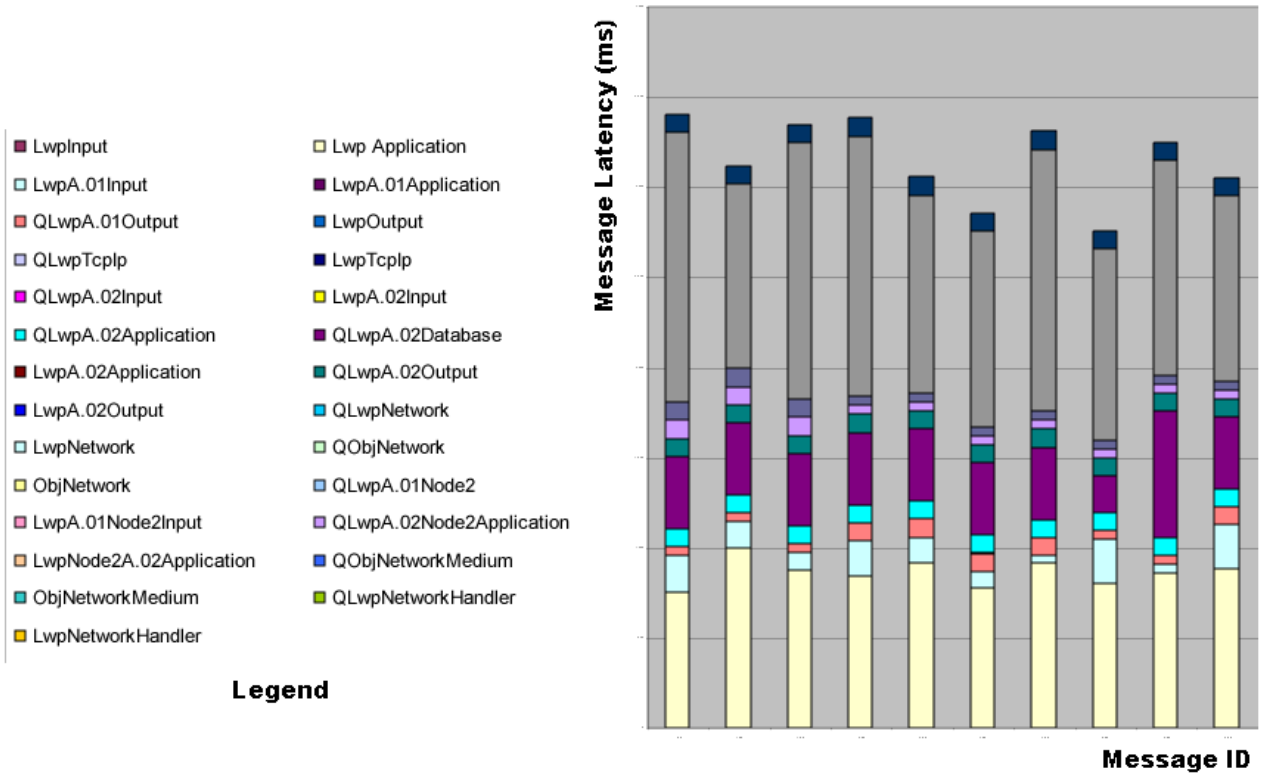

Figure 6: Message Latency Statistical Charts Identify Delays by Component Source

There are a number of challenges yet remaining in this application regarding the design of input and output user interfaces for experimental design and execution. While Silk has no output processing capabilities within the core API, Java database connectivity is being investigated to leverage spreadsheet and database interfaces for auto-mating output analysis and presentation. The advantage of using an industry-standard programming language does mean that third-party Java tools for these tasks are available and must simply be accessed from within the Silk simulation classes. The downside is that this additional non-simulation specific code may reduce the ability to reuse simulation objects in other applications where these optional input and output objects are not required.

REFERENCES

Banga, G. and P. Druschel. 1997. Measuring the capacity of a web sServer, USENIX Symposium on Internet Technologies and Systems (SINTS).

George, Alan D., R. Fogarty, J. Markwell and M. Miars. 1999. An integrated simulation environment for parallel and distributed system prototyping, *Simulation* 72(5): 283-294.

Healy, K. and R. Kilgore. 1997. Silk™: A Java-based process simulation language. In *Proceedings of the 1997 Winter Simulation Conference*, ed. S. Andradóttir, K. Healy, D. Withers, and B.L. Nelson, 475-482. Institute of Electrical and Electronics Engineers, Piscataway, New Jersey.

Healy, K. and R. Kilgore. 1998. Introduction to Silk™ amd Java-based simulation. *In Proceedings of the 1998 Winter Simulation Conference*, ed. D. Meideros, E. Watson, J. Carson, and M. Manivannan, 327-334. Institute of Electrical and Electronics Engineers, Piscataway, New Jersey..

Hu, James, Sumedh Mungee and Douglas C. Schmidt. 1998. Principles for developing and measuring high-performance web servers over ATM, In *Proceedings of INFOCOM '98*.

Joines, J.A. and S. D. Roberts. 1996. Design of object-oriented simulations in C++. In *Proceedings of the 1996 Winter Simulation Conference*, ed., John Charnes, Douglas Morrice, Dan Brunner, and James Swain, 65-72. Institute of Electrical and Electronics Engineers, Piscataway, New Jersey.

Kilgore, R. A., Healy, K. J. and Kleindorfer, G. B. 1998. The Future of Java-based Simulation. In *Proceedings of the 1998 Winter Simulation Conference*, ed. D. J. Medeiros, E. F. Watson, J. S. Carson, M. S. Manivannan, 1707-1712. Institute of Electrical and Electronics Engineers, Piscataway, New Jersey.

Kilgore, R. and K. Healy. 1998. Java, Enterprise simulation and the Silk™ simulation language. In

Proceedings of the 1998 International Conference on Web-Based Modeling & Simulation, ed. P. Fishwick, D. Hill, and R. Smith. SCS, San Diego CA.

Kilgore, R., K. Healy, and G. Kleindorfer . 1998. Silk™: Usable and reusable Java-based object-oriented simulation. In *Proceedings of the 12th European Simulation Multiconference*. SCS International, Ghent, Belgium.

Pidd, Michael , Noelia Oses and Roger J. Brooks. Component-based simulation on the Web? 1999 In *Proceedings of the 1999 Winter Simulation Conference,* ed., P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans, Institute of Electrical and Electronics Engineers, Piscataway, New Jersey.

Rational Software, 2000. UML (Unified modeling lLanguage), <www.rational.com/um>.

ThreadTec, Inc. 1997. Silk®, A Java-Based Process Simulation Language, <www.threadtec.com>.

AUTHOR BIOGRAPHIES

RICHARD A. KILGORE is President of ThreadTec, Inc., the developers and distributors of the Silk simulation software. He has over 20 years of experience as a modeling consultant to Fortune 500 firms in a variety of industries. He received his B.B.A. and M.B.A degrees from Ohio University and Ph.D. in Management Science from the Pennsylvania State University. Formerly, he was a capacity-planning analyst with Ford Motor Co. and Vice-President of Products for Systems Modeling Corp. His e-mail and web address are <kilgore@threadtec.com> <www.threadtec.com>.

EMMETT BURKE is a consultant in the design and optimization of distributed systems architectures. He has over 20 years of experience with a number of advanced design teams in the analysis and performance measurement of a wide range of commercial and military distributed systems projects. His interests include object-oriented simulation and the application of UML for system design and documentation. He is also involved in the design and optimization of advanced technologies used in wastewater treatment facilities. . His e-mail address is <emmett@ symbi.com>.