ISSUES IN PARALLEL DISCRETE EVENT SIMULATION FOR AN INTERNET TELEPHONY CALL SIGNALING PROTOCOL

Phillip M. Dickens
Vijay K. Gurbani

Department of Computer Science and Applied Mathematics
Illinois Institute of Technology
Chicago, Illinois
Email: pmd@work.csam.iit.edu, vkg@rice.iit.edu

Paper code: S262

KEYWORDS: PDES, INTERNET, TELEPHONY, JAVA, FEATURES, SOFTSWITCH

ABSTRACT

In the world of Internet telephony, features are implemented somewhat differently than they are in the existing circuit-switched world. In Internet telephony, features can be implemented as Java applets that are controlled by an Internet call controller. As call signaling and feature deployment move towards the Internet as the network of choice, there is an increasing need to understand and predict the behavior of such systems in the Internet environment. One way to provide such insight is to develop discrete event simulation models of the entities involved in signaling and feature deployment. Given the size and complexity of such models, it is necessary to distribute the simulation across multiple computers to achieve reasonable performance.

In this paper, we discuss the issues involved in designing and implementing a parallel discrete event simulator for Internet telephony. In particular, we look at the issue of how to model network traffic contention within the Internet, how to best model the Java applets implementing the user features, how to synchronize the simulation, and how to maximize the performance of the simulation. While we discuss these issues within the context of Internet telephony, the solutions will be applicable to a wide range of simulation problems.

1 INTRODUCTION

In the world of Internet telephony, features[1] are implemented somewhat differently than they are in the existing circuit-switched telephony world. In the circuit-switched world, features are provided mostly by originating and terminating switches (it's actually a little bit more complicated than this since the features may not reside on the switch itself, but another host accessible to the switch; however, for our discussion, this abstraction will suffice). When a caller picks up a phone and dials a callee, the originating switch applies features to the caller's end point. When the call reaches the terminating switch, the callee's features are applied to the call.

In Internet telephony, with its emphasis on speed of creating and deploying features, the model of applying call features is somewhat different. In the circuit-switched world, a feature is written in special, often proprietary languages and, out of necessity, tied to a switch. In Internet telephony, a feature may well be a Java applet, spawned by an Internet call controller, and running on behalf of some user X. As an example, the feature being implemented may keep track of which of X's friends visit a certain web site, and upon her visit, invite her to a phone conference with X. The invited friend may herself have a feature applet to field such requests and deal with them appropriately. For instance, if it is between 9:00 AM and 12:00 PM, it is okay for X to call. Otherwise, X's call will be directed to her voicemail system.

User feature applets, once started, communicate between themselves using the Internet as the transmission medium and a call processing state machine to keep the call state

[1] A feature is a service provided by the telephone network; thus call waiting and caller ID are examples of telephony features.

synchronized between them. The call processing state machine is thus distributed by its very nature.

We are in the process of designing and implementing a distributed discrete event simulator to simulate the interactions of feature applets during the call setup phase, as they synchronize themselves based on the distributed call processing state machine. The simulator will answer the following basic question: given N users in a call, each with a feature applet, and the network throughput of the transmission medium being P ($0 <= P <= 1$) how long does it take to establish a call between the parties? It turns out however that developing such a simulator raises many interesting and difficult issues that have not yet been addressed by the simulation community. In this paper, we describe these issues and discuss possible solutions.

The rest of this paper is organized as follows: Section 2 gives a detailed look into the entities that comprise the call controller and feature applets, and the messages passed between them to successfully execute a user feature. Section 3 outlines the major issues to be addressed as this research moves forward, and Section 4 outlines related work.

2 ENTITIES INVOLVED IN THE SIMULATION

The Q.931 protocol [Itu98] is one of the primary UNI (User-to-Network Interface) protocols, and is the protocol used in our simulations. We begin by identifying the primary entities involved in the simulation, and the Q.931-like messages (signals) that are passed between these entities to establish a call.

The main entities in our simulation are:

1. The Internet softswitch running the call controller (coordinates the call). The softswitch is assumed to be executing on a general-purpose workstation or high-performance PC.
2. The device associated with the caller (phone, PC, etc.).
3. The user feature applet associated with the caller.
4. The device associated with the callee (phone, PC, etc.).
5. The user feature applet associated with the callee.

Note that there can be multiple callees participating in a call, each with its own device and user feature applet. However, there is only one caller (with its associated device and user feature applet).

Callers initiate the call; callees are recipients of the call. As the call signaling progresses, the devices communicate with the Internet call controller to inform it of certain events, which in turn causes the Internet call controller to send messages to the caller's and callee's user feature applets. The Internet call controller also sends messages to the devices to inform them of certain events. The set of messages a device sends to and receives from the Internet call controller consists of the following:

- **Call Request**: *Sent* by a caller's device when the user owning it wants to make a call. *Received* by the callee's device to indicate a call request has been initiated.
- **Call Proceeding**: *Sent* by the callee's device when call establishment procedures have been initiated. *Received* by the caller's device to indicate that call establishment procedures have been initiated.
- **Call Alerting**: *Sent* by the callee's device to indicate that the callee has been "alerted." *Received* by the caller's device
- **Call Connect**: *Sent* by the callee's device when the callee picks up her endpoint. *Received* by the caller's device to indicate that the callee has connected.
- **Call Disconnect**: sent by either callee or caller's device when the party hangs up.
- **Call Disconnect Ack** Sent by the party that receives the **Call Disconnect** message.

As noted, the user feature applet also receives messages from the Internet call controller. These messages are sent as the call state progresses and result in the execution of certain callback methods in the receiving applet. Note that it is also possible for user feature applets to send messages to each other through the Internet call controller. The callback methods executed by an applet in response to messages sent by the call controller include:

- **onInitCaller:** The caller's applet is being initialized.
- **onInitCallee:** The callee's applet is being initialized.
- **onCallProceeding:** Call is being sent to the callee.
- **onCallAlerting:** The callee is being alerted; callee hears audible ringing, for instance.
- **onCallConnect:** Callee has picked up her end device (i.e. telephone).
- **onCallDisconnect:** Callee or caller has hung up.
- **onCallDisconnectAck:** Call has been released.
- **onSendCallProceeding:** propagates a Call Proceeding message to all other participants in the call, except the participant who generated the Call Proceeding message.
- **onSendCallAlerting:** propagates a Call Alerting message to all other participants in the call, except the participant who generated the Call Alerting message.
- **onSendCallConnect:** propagates a Call Connect message to all other participants in the call, except the participant who generated the Call Connect message.
- **onSendCallDisconnect:** propagates a Call Disconnect message to all other participants in the call, except the participant who generated the Call Disconnect message.

Figure 1 depicts a time-line diagram representing the flow of messages for call establishment that occur between the caller's user feature applet, the Internet call controller, and the callee's user feature applet. The caller picks up her device and initiates a call, which results in a **Call Request** message to be transmitted to the call controller from the caller's device. The call controller then loads the caller's and the callee's user feature applets in memory and calls two callback methods: **onCallerInit()** for the caller and **onCalleeInit()** for the callee. The original **Call Request** message then makes its way to the callee's device, notifying it of the caller's desire to set up communication.

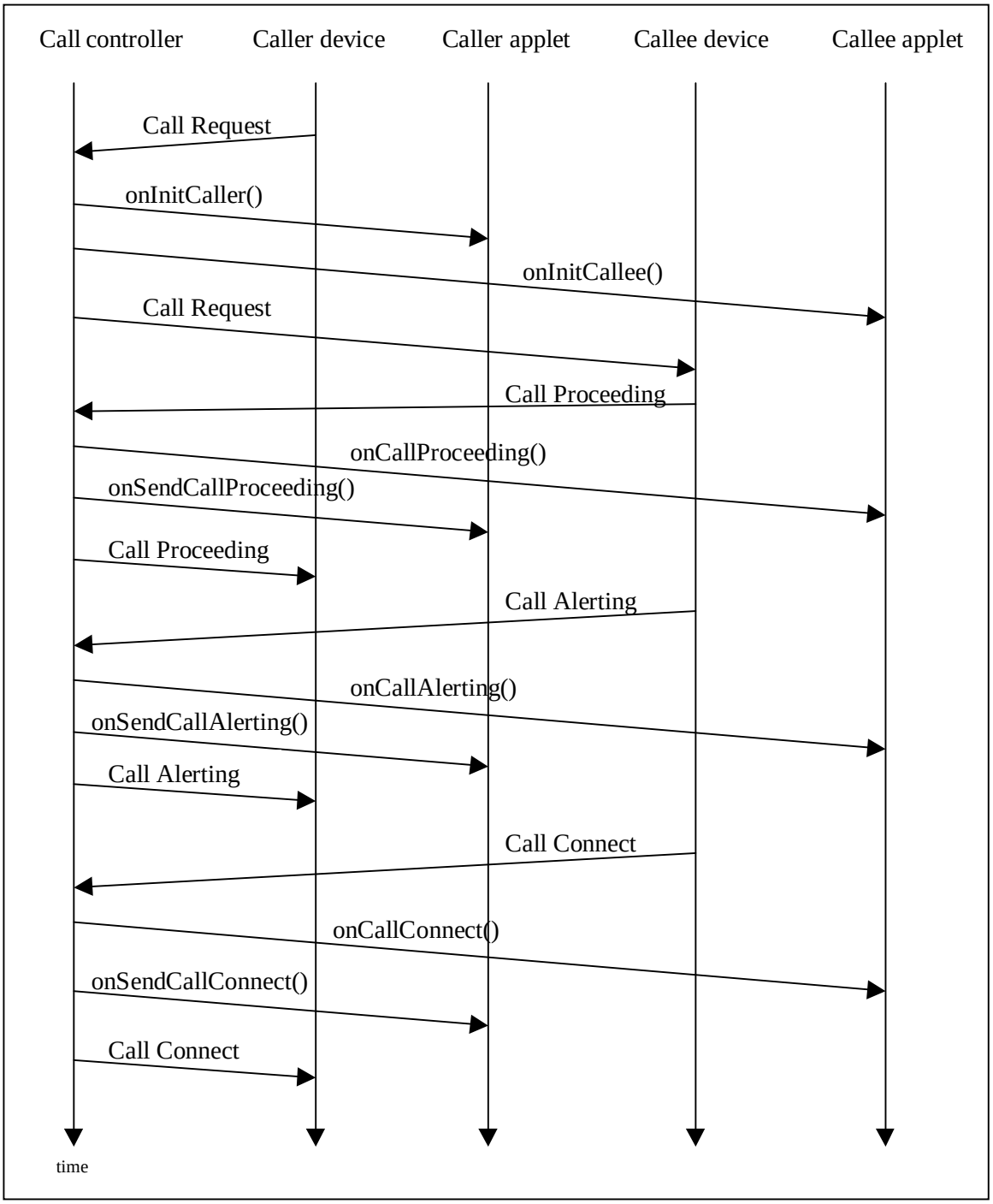

Figure 1: Call establishment messages and callbacks.

The callee's device, upon receiving the **Call Request** message, sends a **Call Proceeding** message to the call controller. This triggers the call controller to invoke two callback methods: **onCallProceeding()** and **onSendCallProceeding()** on the callee and caller respectively. The call controller then forwards the original **Call Proceeding** message from the callee's device to the caller's device.
Following the **Call Proceeding** message, the callee's device sends a **Call Alerting** message to the controller, which in turn invokes two callback methods: **onCallAlerting()** and **onSendCallAlerting()** on the callee and caller respectively. As before, the call controller then forwards the original **Call Alerting** message from the callee's device to the caller's device. When the callee's device gets the **Call Alerting** message, it emits an audible (or visible) indicator to the callee. As soon as the callee picks up its end point, a **Call Connect** message is sent from the callee's device to the call controller. The call controller, upon receiving the **Call Connect** message sends out an **onCallConnect()** and **onSendCallConnect()** to the caller and callee respectively. Once the **Call Connect** message makes its way to the caller's device, the call is established and voice data can begin to flow through the medium (the Internet in this case).

Eventually, one of the parties will hang up their end of the connection. When that happens, the call controller propagates disconnection events to the participating applets and devices, much as it did the connection-related events. We do not discuss further the call disconnect events.

3 ISSUES

There are a number of interesting issues that arise when designing a parallel discrete event simulation to model this system. Clearly, there is a significant amount of messaging activity that has to take place to perform this signaling, and this raises the question of how to best model Internet traffic contention within the simulation. Another issue has to do with the best way to model the activity of the user feature applets. One approach would be to model the applets by actually executing them and observing their behavior. Another approach would be to model the applet's execution with a random (or constant) completion time, and a set of predefined actions caused by its execution. The choice of implementation language is also an important consideration that will have a broad impact on the performance and scalability of the simulation. Finally, since the simulation model is executing across multiple platforms, there must be some mechanism to ensure simulation fidelity. The choice of this synchronization mechanism will have a significant impact on the performance and scalability of the simulation. In this section, we discuss each of these issues.

3.1 Modeling Network Contention

Consider the case of a caller's device sending a Call Request message to the call controller. It is clear that a certain amount of delay, say **t**, will be encountered by this message as it traverses the Internet from the caller's device to the call controller. This delay is clearly a function of the amount and type of Internet/Intranet traffic at that point in time plus the processing overhead at each of the hosts. So how do we quantify **t**? There are three possible approaches:

1. Write our own Internet/Intranet model as part of the simulation model. The problem with this approach is that such a model is extremely complex (and would thus make an already complicated simulation model even more complicated), and that developing such a model from scratch would duplicate a significant body of work aimed at developing scalable Internet simulation models (e.g. SSF[Cowi99]).
2. Another approach would be to build a model of the Intranet (or parts of the Internet) of interest using a pre-existing simulation package (such as SSF or ns [NS]) and execute the network simulation *offline* under various loads and configurations. This would provide some set of values for **t** that could be plugged into the simulation (for example, a value for **t** under light, medium and heavy loads). The disadvantage of this approach is that it is less flexible than simulating the network contention as the call signaling simulation is executing. In particular, it does not allow for the interaction of the call signaling simulation and the network simulation (e.g. in the real system the voice packets generated during a telephone call have an impact on the level of traffic on the common network medium).
3. The best approach, albeit the most difficult, would be to incorporate into our simulation a pre-existing simulation framework that simulates the Internet/Intranet traffic throughput. In this case, our simulator would input the call signaling message traffic into the external framework, and simply observe the time at which it reaches its destination. In effect, our simulator would be just another traffic source for the external framework.

   For example, consider Figure 2. In this example, the device associated with user 1 sends a Call Request message to the call controller at time $\mathbf{t_0}$. Thus this message enters into the network (being simulated by a an external simulation framework) at time $\mathbf{t_0}$. The message then competes for network bandwidth with the other traffic being modeled in the external framework (and any other call signaling messages already in the network). In this example, the Call Request message

exits the external simulation framework (and arrives at the call controller) at time $\mathbf{t_1}$.

This approach of course requires an external simulation framework with well-defined inputs for our simulator to plug into. We are currently investigating existing simulation frameworks to determine if they can be used in this manner.

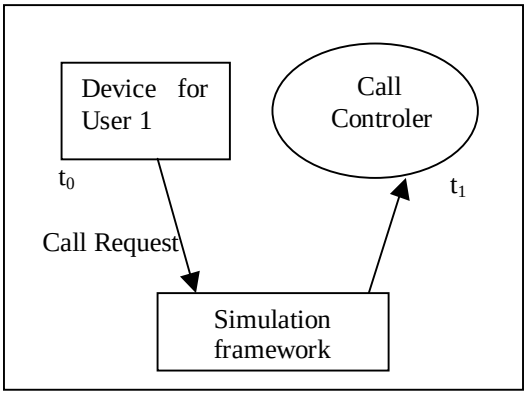

Figure 2: A simulation framework.

3.2 Direct execution simulation or "educated guess?"

A key component of the next generation Internet telephony will be the user feature applets that can be applied to the calls as they are being set up. Given the importance of this aspect of the system, it is critical to realistically model the feature applets in the simulation. So what is the best way to model such features? There are two basic options that we discuss in turn.

One way to model the user feature applets is to use the technique of direct *execution simulation.* In this approach, the behavior of a user feature applet is obtained by actually executing the applet and observing its behavior. Thus the call controller will actually spawn Java Virtual Machines (JVMs) when initializing the caller and callee applets. Subsequently, when a callback occurs the JVM previously spawned actually executes the callback method to determine its behavior. This of course provides the simulation with a very accurate model of the behavior of the applet being executed.

There are however difficult issues that must be addressed when using direct execution simulation. Perhaps the most important issue is how to determine the length of time it takes to execute a user feature applet. The problem presents itself when there are not enough computers available to dedicate a CPU to *each* of the user applets executing at a particular time in the simulation. To understand this issue, it is important to note that when an applet has a dedicated machine, hardware clocks can be used to provide an accurate accounting of the time the applet executes. When multiple applets are being multiplexed on the same CPU however, hardware clocks cannot provide this same service. The problem is that such clocks do not account for the time a process is swapped out due to the effects of multiprogramming. The issue then is how to factor out the length of time an applet is swapped out due to the effects of multiprogramming. (The interested reader is directed to [Dick96] for a detailed discussion of this problem.)

One solution to this problem would be to develop a model of the effects of multiprogramming, and to use this model to determine the amount of time an applet would have executed had it not been for the effects of multiprogramming. This certainly complicates the simulation model, but the more accurate picture of feature applet behavior may be worth the additional complexity.

The second approach to modeling the behavior of the user feature applets could be considered the "educated guess" scenario. In this approach, some reasonable estimate of applet execution time is developed and plugged into the simulation model. In a similar manner, some reasonable estimate of the behavior of the applets would have to be included in the model.

3.3 Implementation Language

We have chosen Java as the implementation language for a number of reasons. First, it has excellent support for distributed computing (including networking and serializing support). Also, Java supports multiple execution threads and handles the details of garbage collection. Perhaps the most important reason to use Java is that it is a highly portable language. In fact, previous research has demonstrated that Java's capabilities match very well with the requirements of distributed simulations [Kilg98, Pidd98].

The disadvantage of Java is that it is notoriously slow. However, as Java has grown in popularity its performance has been improving. An important aspect of this research will be the development of mechanisms to optimize Java's performance in this arena.

3.4 Synchronization protocol

The problem of maintaining temporal or causal fidelity between the various entities in a PDES is well studied [Fuji90]. The solution to this problem can be approached from two angles: using "conservative" techniques, or using "optimistic" techniques.

The "conservative" techniques work by ensuring that when a host processor processes an event with timestamp *t*, it will never receive an event with timestamp less then *t* from any other processor in the simulation [Lege96]. It has

been well established that for conservative techniques to work well, good *lookahead* (defined as the simulator's ability to predict its future behavior) is required. Thus if a conservative approach is adopted, an important component of the research will be to identify and exploit the lookahead available in the simulation model.

"Optimistic" techniques, on the other hand, allow a processor to optimistically execute any event it receives (without waiting to ensure that it will never receive an event with a smaller timestamp). A state saving and rollback mechanism is employed to correct any out-of-order processing that does occur.

The decision to use either a conservative or an optimistic synchronization technique is still being investigated. Either of these techniques can be applied to the simulation being proposed, and we will perform some preliminary studies to determine which approach appears to be the most promising.

4 RELATED WORK

The work most closely related to this project is the call signaling simulation described in [Eyer00]. The primary difference is that their work pays close attention to the signaling aspect and does not consider the modeling of user feature applets, which is a significant component of this project.

[Lin95] studied PDES and its application to a specific field in telecommunication – Personal Communication Service (PCS). The goal of the PCS simulation is to optimally design cellular networks to minimize the blocking probability (a call is blocked if there are no channels available at a given cell). [Lin95] describes both conservative and optimistic simulations for PCS.

Other related work includes SSF[Cowi99] and ns[NS], both of which are network simulation packages. One of our goals is to incorporate the services of such packages in our simulation framework.

REFERENCES

[Cowi99] Jim Cowie, Hongbo Liu, et. al. Towards Realistic Million-Node Internet Simulations. *Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications*, http://www.ssfnet.org/Papers/pdpta99.pdf.

[Dick96] P. Dickens, P. Heidelberger and D. Nicol. Parallelized Direct Execution Simulation of Message-Passing Parallel Programs. In *IEEE Transactions on Parallel and Distributed Systems.* Zvolume 7, Number 10, October 1996.

[Eyer00]Tony Eyers and Henning Schulzrinne. Predicting Internet Telephony Call Setup Delay. http://www.cs.columbia.edu/~hgs/papers/Eyer0004_Predicting.pdf

[Fuji90] Richard M. Fujimoto. Parallel discrete event simulation. *Communication of the ACM*, 33(10):30-53, October 1990.

[Itu98] ITU-T Recommendation Q.931 (1998) *ISDN user-network interface layer 3 specification for basic call control*.

[Kilg98] Richard A. Kilgore, Kevin J. Healy and George B. Kleindorfer. The future of Java-based simulation. In *Proceedings of the 1998 Winter Simulation Conference*.

[Lege96] Ulana Legedza and William E. Weihl. Reducing synchronization overhead in parallel simulation. *1996 Proceedings of the 10th Workshop on PADS*, pages 86-95.

[Lin95] Yi-Bing Lin and Paul A. Fishwick. Asynchronous Parallel Discrete Event Simulation. In *IEEE Transactions on Systems, Man, and Cybernetics, 1995.*

[NS] UCB/LBNL/VINT Network Simulator – ns (version 2), http://www-mash.cs.berkeley.edu/ns/

[Pidd98] M. Pidd and R.A. Cassel. Three phase simulation in Java. In *Proceedings of the 1998 Winter Simulation Conference.*

[Rapp96] Theodore S. Rappaport. Wirless Communications: Principles and Practices. *PrenticeHall Publishing Company, 1996.*

[Schu99] Henning Schulzrinne, Mark Handley, Eve Schooler, and Jonathan Rosenberg. SIP: Session Initiation Protocol. IETF RFC 2543.