Proceedings of the 1997 Winter Simulation Conference
ed. S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson

JAVA BASED CONSERVATIVE DISTRIBUTED SIMULATION

Alois Ferscha
Michael Richter

Institut für Angewandte Informatik
Universität Wien
Lenaugasse 2/8, A-1080 Vienna, AUSTRIA

ABSTRACT

Distributed discrete event simulation techniques aim at an acceleration of the execution of a self-contained simulation model by the spatial decomposition of that model and the concurrent simulation of the submodels by so called *logical processes* (LPs), each executing on a dedicated node of a (closed) multiprocessor system. The dedication of parallel simulators to specific platforms and their adaptation to the respective hardware and software intrinsics has widely preventend an industrial and/or commercial success of such high performance simulation methods.

In this work we propose the building of logical process simulators that use the World Wide Web as an execution platform. We have developed and implemented a Java-based simulation engine along the conservative Chandy/Misra/Bryant (CMB) synchronization protocol, allowing for a platform independent, scalable, and performance efficient distributed simulation model execution. Our simulation engine is open and general purpose in the sense that it can be reused in various simulation domains by simply "plugging in" different simulation models, and simulation codes that, once written, can be executed anywhere (on the Internet). The potential performance improvement of Web-based distributed simulation is investigated in a sensitivity analysis conducted on a hypothetical simulation model executed in a Java-enabled workstation LAN. Even for very small simulation models, a speedup of about 3.5 could be attained on a 4 processor (heterogeneous) host LAN.

1 INTRODUCTION

Over its more than 15 years of existence, the field of parallel and distributed discrete event simulation has developed a broad body of theory and methods after the pioneering works (Chandy, Misra 1979) and (Jefferson 1985) (see Ferscha 1996 for a recent survey). Nevertheless, it has suffered from simulationists and industrials being reserved on the potential gains of these methods, considering the complexity of development and implementation efforts. This failure in generating general acceptance in simulation practice is one of the main reasons for a recent existential discussion on the chances of survival of parallel and distributed simulation as reflected in a collection of papers following Fujimoto's position statement in (Fujimoto 1993).

Only few attempts to escape from this pessimistic image have appeared in the recent past, among which the ones integrating parallel and distributed simulation techniques into commercial/industrial simulators in a transparent way (Nicol, Heidelberger 1995) are the most succesful. We also believe that the transparent use of high performance simulation techniques is the key to success in simulation practice, but see an even better chance for a breakthrough in the use of the World Wide Web and mobile code as the enabling technology for distributed simulation. In this sense, this paper is in the spirit of Nicol and Heidelberger: *"As the mountain is not coming to the Prophet, evidently the Prophet must go to the mountain."* (Nicol, Heidelberger 1995), but attacks transparency in a more consequent way.

In this work we first point out relevant issues of "transparency" in heterogeneous environments, and isolate Web technologies that – at the time being – already provide key features for a transparent use of the WWW as a distributed simulation platform. We collect arguments as to why Java embodies key success factors towards this goal. In Section 3 we explain the architecture of a Java enabled conservative distributed simulation engine, its universal portability, and its generality with respect to the execution of different simulation models. Section 4 is devoted to a case study. A Java thread implementing a conservative LP simulation engine is developed, and RMI is involved for LP synchronization. Experiments demonstrate that considerable simulation execution acceleration can be achieved.

2 TRANSPARENT DISTRIBUTED SIMULATION ON THE WWW

The homogeneous and centralized *parallel processing systems* which have been the subject of investigation in the context of parallel and distributed simulation are nowadays widely being replaced by heterogeneous, distributed systems. Instead of an isolated CPU pool in classical parallel processing systems, the "network" itself appears as a high performance computing platform, in which the collection of network devices constitutes the pool of resources (CPUs, storage, communication devices). As an example, today's WWW appears as a vast "pool" of such resources, readily available (besides other things) for **World Wide Simulation** (WWS). (Some thoughts on the use of the Web for simulation are collected in Fishwick 1996.)

A key problem towards the use of the WWW as a platform for distributed simulation model execution is *transparency*, i.e. the problem of **transparent access** to net resources. The enabling technologies considered here for making the different kinds of heterogeneity transparent to users are the following:

- *transparency of network heterogeneity*: Basic Web-technology has solved the problem of interoperability of different networks based on well-defined, standardized protocols like HTTP and CGI. The HTTP protocol establishes a uniform information transport mechanism, whereas CGI defines in a clear way the interface between a Web-server and arbitrary programs executing on the same physical machine. The hypertext markup language HTML defines the syntax and logical structure of arbitrary content, and resources are referenced across heterogeneous networks by a global naming scheme, i.e. the Uniform Resource Locator (URL).

- *transparency of platform and operating-system heterogeneity*: The Java virtual machine provides a uniform processing platform and is well established due to its integration into standard Web-browsers, available on all major hardware platforms. The Java compiler translates class-based object-oriented (Java) source code to an intermediate machine-independent and platform-independent byte-code, which is interpreted by the Java virtual machine, a platform-neutral architecture definition. Only the virtual machine has to be ported to a specific platform and operating system. In the context of simulation, this has already influenced some exploitations: Sequential discrete event simulation with Java is investigated in (Buss, Stork 1996), the development of a Java simulation library is reported in (Nair, Miller, Zhang 1996).

- *transparency of user-interface heterogeneity*: Along with the Java programming language, a class library for API programming and user interface programming (AWT) has emerged, supporting standardized concepts for graphical interfaces (GUIs). An additional and innovative feature is the possibility of run-time linking, enabling dynamic retrieval of classes from the net (e.g. loading applets embedded in HTML documents). Extensive reuse and the combination of existing GUIs and software aims at cutting down user interface development efforts, while at the same time preserving cross-platform and cross-operating system transparency.

Accomodating these basic needs for transparency, contemporary Web technology provides an excellent basis for Web based distributed simulation. Moreover, all this technology is right at hand today, at almost any place in the world. By using Java technology for the realization of distributed simulations, a universal portability (and platform independence) of distributed simulations can be guaranteed. Further chances for a breakthrough succcess can be seen in the following key features of the Java technology:

- The utilization of simulation software resources from anywhere in the Internet, from any machine and independent of the local operating system (ease of access to world wide computing resources).

- The utilization of simulation models from anywhere in the Internet. This feature highly promotes simulation model reuse/sharing: (sub) models (like simulation software) can be created and posted to a Web site and used as "plug-ins" in various different simulations.

- The estalishment of a world wide model / simulation software base according to the previous features.

- The support for and simplification of simulation code migration from one system node to another, and the instantiation of local simulation code on remote system nodes (e.g. remote method invocation). This feature adds a new dimension of dynamicity to contemporary simulation methods.

- The reuse of existing simulation code in a transparent way, without any rewriting, without any

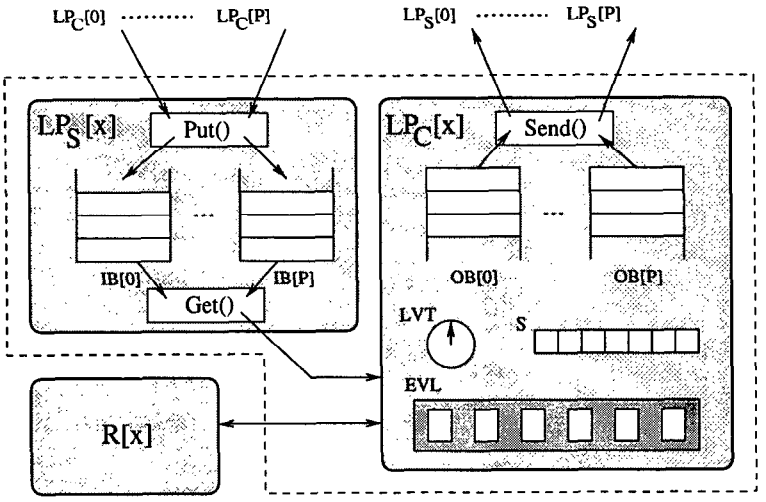

Figure 1: Architecture of a Conservative Logical Process

recompilation (*"write-once run-anywhere"*). A geographically dislocated code and model development is supported, e.g. in "cells of expertise".

- The scalability of distributed simulations: simulation code can be instantiated (remote method invocation) to any desirable extent of multiplicity, since physical execution devices (CPUs) are available to an almost infinite extent.
- Among other simulation language candidates (C, C++, etc.) Java is the only language that is inherently "Internet-aware", easy to learn, modular, supports processes and process interactions (multithreading), and eases the integration of simulations into intuitive, graphical and animated user interfaces.
- Last, but not least, distributed simulations can be performed at anytime from anywhere.

Having motivated simulation modeling and execution based on Web and Java technologies, we now build a distributed simulation engine within this framework, and exemplify the feasibility and enhanced quality of "world wide simulations."

3 A JAVA BASED, CONSERVATIVE DISTRIBUTED DISCRETE EVENT SIMULATION ENGINE

An object oriented distributed discrete event simulation engine using the Chandy-Misra-Bryant (CMB) protocol (Chandy, Misra 1979), (Bryant 1984) and the Java RMI (remote method invocation) system is developed in this section. In order to accelerate the execution of a self-contained simulation model, the CMB approach to distributed simulation decomposes the model into spatial submodels, each of which is executed by so called *logical processes* (LPs). To preserve intra-LP event causality during simulation, CMB protocols (as opposed to Time Warp protocols) execute in each LP events that occur in the respective submodel in nondecreasing order of their occurrence timestamp, thus strictly preventing the possibility of any event causality violation across LPs. For this, CMB protocols rely on the detection of when it is (causally) *safe* to process an event (i.e. detect so called lookahead). A deadlock management mechanism is necessary to either avoid deadlocks (due to cyclic waiting conditions for messages able to make 'unsafe' events safe), or detect and recover from deadlocks. In distributed system implementations of CMB protocols deadlock management is usually based on message exchanges. The architecture of a CMB LP must thus cover three different functionalities: (*i*) an input-sided communication interface taking care of arriving messages (input buffer, IB) (*ii*) a simulation engine that (under the control of the input communication interface) advances local simulation time LVT by executing events scheduled in the event list EVL and adjusting the local state variable vector S respectively, and (*iii*) an output-sided communication interface involved in propagating the effects of the local execution in the form of events to remote LPs as messages. (See e.g. Ferscha 1996 for a very detailed explanation of the architecture of a CMB simulator).

In our Java implementation of an LP adhering to CMB, the architecture is viewed as containing two parts (Figure 1): the input interface (IBs) managed by the Java class LP_S, and the simulation engine and output interface as managed by the Java class LP_C.

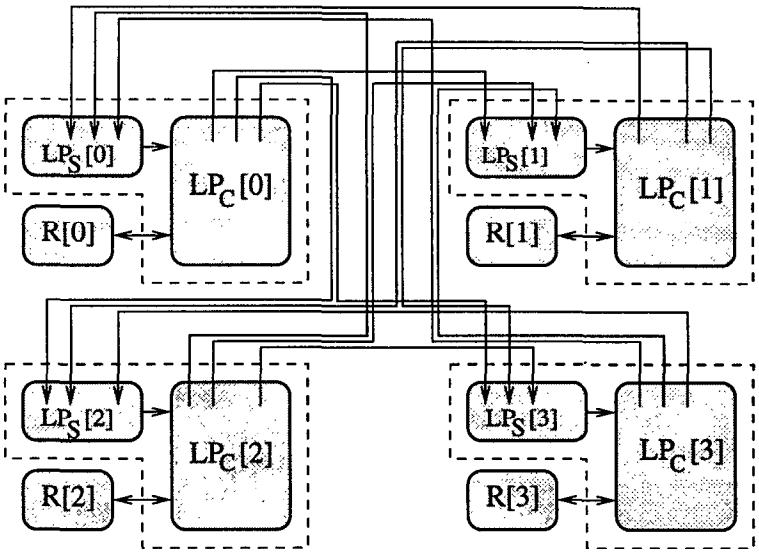

Figure 2: Possible RMI Using 4 LPs

Each $LP_C[i]$ executing an LP_i accesses a dedicated simulation submodel (or spatial region of the simulation task) R. The Put-method of the LP_S class inserts arriving messages in the corresponding IB and can be invoked from every $LP_C[j]$ $(j \neq i)$ involved in the federated simulation, whereas the Get-method can be invoked only locally from $LP_S[i]$, i.e. at LP_i. In this way, LP_Cs can send messages to all LP_S IBs, but will receive messages only from its own (local) IBs. Every LP_C can request the smallest time stamp of all IBs of its LP_S, compute the minimum timestamp of messages and determine the time horizon (local virtual time horizon, LVTH) up until which it is safe to execute events from the local EVL. Within the time frame of LVT and LVTH events are processed, yielding new states. If the respective new state enables new events occurring in the local future, these are then scheduled in the EVL. Conversely, if scheduled events have become disabled, they are removed from EVL. Outgoing messages are deposited in the local output buffers ($OB[i]$) together with nullmessages which encode the local lookahead. (Note that a sender-initiated nullmessage policy is used here.) After executing all the safe events, the whole contents of the local output buffer is sent to the input buffers of the corresponding recipient.

Each LP_C executes in its own Java thread, "yielding" control to another LP_C after one simulation cycle. Via this direct scheduling of LPs, the whole simulation is made independent of proprietary scheduling strategies encoded in the different Java virtual machines. The thread based implementation, allows in a natural way, for a model with n LPs to be simulated in a distributed environment with m hosts, as well as even on a single host without any source code modification. Thus a highest possible degree of portability, partitioning and load balancing options are preserved in this way. Figure 3 describes the Java thread that implements a simulation engine SE.

The interconnection of multiple LPs is illustrated in Figure 2. As far as the topology of LP interconnections is concerned, only the connections necessary for the particular simulation model are instantiated at runtime. A link between $LP_C[0]$ and $LP_S[1]$ will therefore only exist if in the submodel R[0] prescribes potential causal effects onto R[1] or vice versa, i.e. R[0] and R[1] will have to exchange messages. (Consequently, and what appears as a particular advantage of the object oriented implementation strategy, the possibility for dynamic LP topologies arives.)

The reason for separating LP_S and LP_C in the implementation is portability to different simulation strategies. In its current form, LP_C implements a CMB protocol which could easily be replaced by a Time Warp simulation engine. The input interface LP_S could be reused for such a new simulator without modification. Even seamlessly switching among different LP_C implementations at runtime appears possible, at least from an implementation viewpoint. Furthermore, the same modularity arguments hold for the isolation of the simulation submodel R: submodels can be replaced by simply "plugging in" a new model class.

4 CASE STUDY

In the following case study we investigate the performance sensitivity of the simulation engine with re-

```
public void SE(){
    MyEvent eh, e;
    e = new MyEvent();
    LVT = 0.0;
    // create initial state
    model.initialMessages(Buffer);
    while (LVT < EndTime){
        // determine local virtual time horizon
        LVTH=Buffer.getIN();
        // test for scheduled, safe events
        if ((EVL.EVLsize()>0)
            && (EVL.ts() <LVTH))
            // choose next internal event
            // and remove from EVL
            e = EVL.remove_first();
        else
            // choose next external event
            // and remove from respective IB
            e = Buffer.remove_first();
        // execute event and advance LVT
        LVT = e.ts();
        if (!e.Null()){ // e is not a nullmessage
            model.modify_by_event(e, coMyID);
            // generate new events, insert in EVL
            model.new_Events(EVL, coMyID);
            // remove preempted events from EVL
            model.remove_Events(EVL, coMyID);
            // outgoing messages to buffer OB[i]
            model.send_Events(Buffer, coMyID);
        }
        //compute lookahead
        Buffer.lookahead(LVT,
            model.lookahead(coMyID));
        //sendout buffer contents
        Buffer.send();
        //deschedule thread
        yield();
    }// endtime reached
    model.SimInfo();
    Buffer.On_Exit();
    this.stop();
}
```

Figure 3: Simulation Engine Thread

spect to the number of LP threads assigned to a particular Java virtual machine, different workloads in the various LPs, the degree of locality of causal effects, and the amount of lookahead imposed by one LP on another. Specifically, the variation of the following factors influencing performance are of interest:

- **Heterogeneous Host**
  The use of hosts with different performance profiles is of interest because this is most probably the situation in a WWS. We therefore intentionally involve a set of four hosts as follows: H1 and H2 (Sparc 5), which are approximately half as fast as H3 (Sparc 20), and H4 (UltraSparc), which is about 3.5 times faster than H1. A distributed simulation involving 4 LPs is executed on all combinations of these hosts.

- **Initial Load Assignment**
  The larger the number of events preliminarily assigned to a region R, the larger the model parallelism, i.e., the number of possible concurrent event executions. However, as the number of events in one region increases, so does the overhead to be managed by the simulation engine, yielding a CPU load increase. We assign almost similar loads (Initial Events) to the LPs as annotated in Figure 4.

- **Interaction among LPs**
  To study performance effects of varying degrees of "locality" in the distributed simulation, we define two different simulation models (Figure 4). In the first case, referred to as Model 1 (Figure 4, left), LPs are linked in a circular way, i.e. event executions in one LP can have causality effects on a single successor LP. The second case, referred to as Model 2 (Figure 4, right), covers the more general situation in which local simulation can affect any remote LP.

- **Lookahead**
  The performace of CMB is highly reliant on lookahead information. In general, the more lookahead information that can be extracted from the simulation model, the higher LVTH in remote LPs enabling them to execute a higher number of events without interference from the outside. We assume the lookahead of an $LP_C[i]$ imposed to another $LP_C[j]$ as being a random variate X following a truncated normal distribution $la_{i-j} = N(l_X, u_X, \mu_X, \sigma_X)$, where l_X and u_X are the lower and upper bounds of possible instancies, and $l_X \leq \mu_X \leq u_X$ and σ_X are the respective mean and variation. For simplicity we assume the same lookahead is imposed on every adjacent LP.

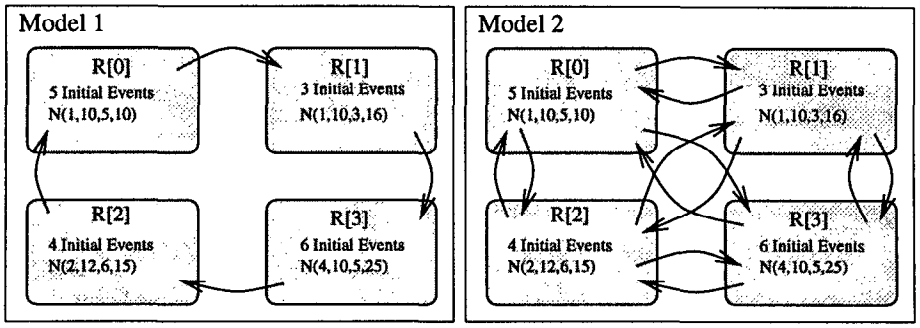

Figure 4: Ring of LPs (Model 1) and Full LP Interconnect (Model 2)

- **Event Grain Size**
  For the CMB protocol it is not necessarily a fast CPU (by itself) that gives good overall performance, nor is it solely the use of high speed communication devices but rather the fitting of the balance of the communication and computation speed to the event structure underlying in the simulation model. In our investigations we control this parameter indirectly via the grain size of events: the CPU time needed to execute one event is assumed to be uniformly distributed over the interval [50ms, 100ms].

- **Message Population**
  The workload in the particular simulation models is designed to keep the message population constant, i.e. each event consumes one message and generates a new one. This is not a limitation by the simulator, shall make performance effects in our experiments more transparent.

- **Message fan out**
  For a generated message, the destination LP is chosen randomly depending on the available output connections.

- **Local Virtual Time Progression**
  For the control of the progression of simulation time in one simulation step, we associate the respective LVT increments with events. Whenever an event is executed, LVT is progressed (LVT = e.ts(); in Figure 3) by a random increment again following a truncated normal distribution $\Delta LVT_C = N(l_C, u_C, \mu_C, \sigma_C)$. (For the two models under investigation ΔLVT_C is notated in Figure 4 as quartuples $N(\_,\_,\_,\_)$.)

The first LP configuration is Model 1 in Figure 4 (left). 4 LPs are interconnected in a ring topology, each executing about the same initial load at a [50ms, 100ms] event granularity. Figure 5 summarizes averaged execution times for the model being mapped to a single, two and four nodes out of H1, H2, H3 and H4. In the case where all LPs are simulated on the same host we obtain average execution times between 314.3 sec (H1,H2) and 89.4 sec (H4) depending on the type of host used. In the cases where 2 hosts share the simulation model (2 LPs are executed on one host) we obtain results between 140.0 sec with the 2 slow host (H1,H2), 99.8 sec if a combination of a slow host and a fast host (H1,H4) are used, and 48.5 sec if the two faster hosts (H3,H4) are involved. In the full exploitation of model parallelism (each node executes one LP), an execution time of 54.9 sec can be achieved. Clearly, in this case, the two slow machines (H1, H2) throttle the performance of the two others.

Figure 5 (right) shows the speedup range gained by the distributed simulation of the model, based on a sequential execution on host H1 or H2 (case "slow"), on host H3 (case "medium") and host H4 (case "fast"). It is seen from these results, that in a heterogeneous environment, depending on the choice of the host system and the LP-to-processor mapping, speedup in the range of 1.62 and 5.84 can be achieved; slight slowdowns are possible as indicated by the variation bars in cases where two processor are used.

The LP configuration in Model 2 (Figure 4 (right)) represents the case of a communication bound distributed simulation. A fully interconnected graph of LPs is assumed, with an equally likely message fanout from one LP to all of its neighbors. The count of initial messages and the LVT advancement is the same as in the previous example. Through the necessity of sending more null messages, the execution time approximately doubles. If all LPs are simulated on one host we obtain execution times between 694.2 sec (H1,H2) and 191.1 sec (H4), and again the host combination H3 and H4 (73.5 sec) is much faster than the combination where all 4 hosts are involved (Figure 6 (left)). The speedup range as presented in Figure 6 (right), however, asserts robustness of the distributed simulation against heavy communication.

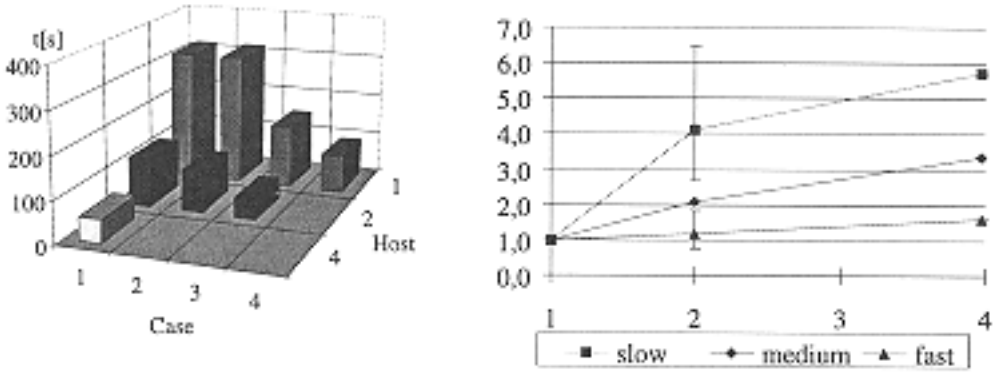

Figure 5: Execution Time (Left) and Speedup (Right) for Model 1

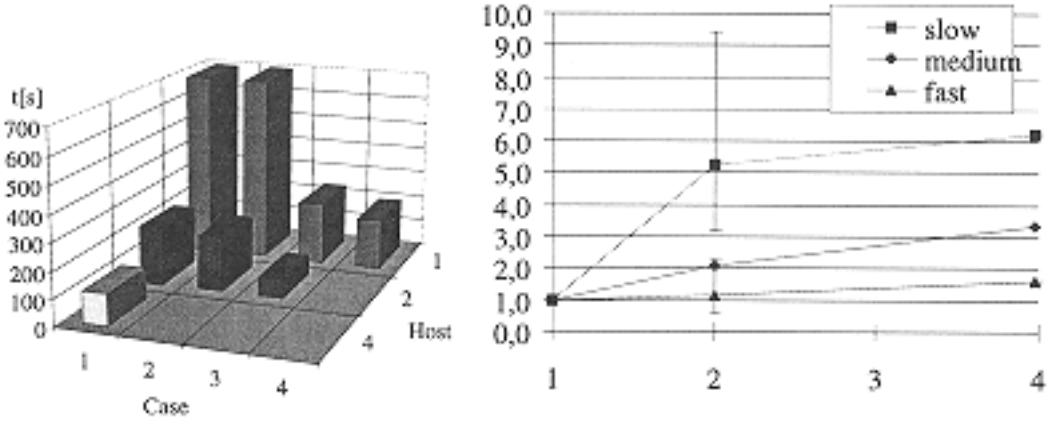

Figure 6: Execution Time (Left) and Speedup (Right) for Model 2

5 CONCLUSIONS

World-wide distributed simulation is closer than expected! In a systematic collection of arguments, this paper has demonstrated that contemporary Web technology provides an excellent basis for Web based distributed simulation. Moreover, using the Java technology for the realization of distributed simulators, a universal portability (and platform independence), modularity, simulation software and model reuse, as well as its use from anywhere at anytime can be guaranteed, thus representing the chance of a breakthrough commercial and industrial success of general purpose distributed simulation.

The feasibility and "value-added" of world wide simulations is exemplified with the development of an object oriented distributed discrete event simulation engine based on the conservative Chandy-Misra-Bryant (CMB) protocol, using the mutlithreading features of Java together with the RMI (remote method invocation) system. The performance of this distributed simulator is investigated in a heterogeneous environment of Java-enabled workstations. Even with a set of four heterogenous hosts, an average speedup of 3.5 could be attained.

Having identified a consistent set of parameters determining the performance of a distributed simulator, we are now studying the performance sensitivity of those. Further experiments will compare the issues of imbalanced initial LP loads, the variation of event grain size, heterogeneous lookahead and virtual time progressions, as well as optimizations of the CMB protocol reducing the number of nullmessages. Finally, we will demonstrate the ease of submodel replacements and the integration of the Time Warp synchronization protocol.

ACKNOWLEDGMENT

This work was partially supported by the Human Capital and Mobility program of the EU under grant CHRX-CT94-0452 (MATCH).

REFERENCES

Bryant, R. E. 1984. A Switch-Level Model and Simulator for MOS Digital Systems. *IEEE Transactions on Computers* 33:160–177.

Buss, A. H., and K. A. Stork. 1996. Discrete Event Simulation on the World Wide Web using Java. In *Proceedings of the 1996 Winter Simulation Conference*, ed. J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain, 780–785.

Chandy, K. M., and J. Misra. 1979. Distributed Simulation: A Case Study in Design and Verification of Distributed Programs. *IEEE Transactions on Software Engineering* 5:440–452.

Ferscha, A. 1996. Parallel and Distributed Simulation of Discrete Event Systems. In *Parallel and Distributed Computing Handbook*, ed. A. Y. Zomaya, 1003–1041. New York: McGraw-Hill.

Fishwick, P. A. 1996. Web-Based Simulation: Some Personal Observations. In *Proceedings of the 1996 Winter Simulation Conference*, ed. J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain, 772–775.

Fujimoto, R. M. 1993. Parallel Discrete Event Simulation: Will the Field Survive? *ORSA Journal of Computing*, 5:218–230.

Jefferson, D. A. 1985. Virtual Time. *ACM Transactions on Programming Languages and Systems*, 7:404–425.

Jefferson, D. R., and H. Sowizral. 1985. Fast Concurrent Simulation Using the Time Warp Mechanism. In *Proceedings of the Workshop on PADS*, 63–69.

Nair, R. S., J. A. Miller, and Z. Zhang. 1996. Java-Based Query Driven Simulation Environment. In *Proceedings of the 1996 Winter Simulation Conference*, ed. J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain, 786–793.

Nicol, D. M., and P. Heidelberger. 1995. On Extending Parallelism to Serial Simulators. In *Proceedings of the 9^{th} Workshop on PADS*, 60–67. Los Alamitos: IEEE CS Press.

AUTHOR BIOGRAPHIES

ALOIS FERSCHA is an Associate Professor for Computer Science at the Department of Applied Computer Systems at the University of Vienna, Austria. He has served as the 1998 Program Chairman and the 1997 General Chairman of the Workshop on Parallel and Distributed Simulation (PADS). His current research interests include performance modeling and prediction, computer aided performance engineering of parallel and distributed software, distributed simulation and distributed cooperative work environments. He is a member of ACM, IEEE and GI.

MICHAEL RICHTER has been a student in the Parallel and Distributed Computing Project Studies at the Department of Applied Computer Systems at the University of Vienna, Austria. His research interests emphasize on software development for distributed, heterogeneous execution environments.