Proceedings of the 1995 Winter Simulation Conference
ed. C. Alexopoulos, K. Kang, W. R. Lilegdon, and D. Goldsman

PARALLEL AND DISTRIBUTED SIMULATION

Richard M. Fujimoto

College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280, U.S.A.

ABSTRACT

Research and development efforts in the parallel and distributed simulation field over the last 15 years has progressed, largely independently, in two separate camps: the largely academic high performance Parallel And Distributed (discrete event) Simulation (PADS) community, and the DoD-centered Distributed Interactive Simulation (DIS) community. This tutorial gives an overview and comparison of work in these two areas, emphasizing issues related to distributed execution where these fields have the most overlap. Differences in the fundamental assumptions routinely used within each community are contrasted, followed by overviews of work in each community.

1 INTRODUCTION

The enormous amounts of computation required by simulations of large, complex systems such as telecommunication networks and VLSI circuits coupled with the widespread availability of multiprocessor computer systems has motivated an extensive amount of research in the execution of discrete event simulation programs on multiprocessors and distributed computing platforms. Research in the parallel and distributed simulation community has been focused on primarily one goal: develop technologies and tools to enable large simulation programs to be developed and executed in as little time as possible to improve the productivity of users of these tools. Here, we collectively refer to this body of work by the *PADS* acronym, named after the annual *Workshop on Parallel and Distributed Simulation* where many of these results appear.

Simultaneously, the needs of the military establishment to have more effective and economical means to train personnel has driven a large body of work in developing virtual environments where geographically distributed hardware and personnel can interact with each other as if they were in actual combat situations. Work in the Distributed Interactive Simulation (DIS) community is now expanding to encompass other uses in the military, e.g., testing and evaluation (T& E) of new weapons systems, as well as commercial applications, e.g., entertainment, training air traffic controllers, and emergency planning to prepare for earthquakes or other disasters. Training remains a central application of DIS environments today. In contrast to PADS research, the principal goal of DIS work has historically been to realize realistic virtual environments utilized by geographically distributed personnel.

Both PADS and DIS model the system under investigation as a collection of entities that interact with each other and their simulated environment in some fashion. In PADS, the entities are almost always computer representations (data structures and code) of queues, tanks, vehicles, etc. DIS simulations integrate *virtual entities* (human in the loop training simulators of, for example, tanks or aircraft), *live entities* (operational manned vehicles and weapons systems), and *constructive simulations* (typically, wargaming simulations that model combat elements at a more aggregated level, e.g., battalions as opposed to individual soldiers) to produce a virtual battlefield. DIS is a simulation infrastructure intended to support interoperability among separately developed simulators, systems, and human participants.

Historically, research in the PADS and DIS communities has proceeded largely independently of each other, and entirely different approaches have been adopted by each. This is understandable given the different perspectives and goals of the two communities. However, as both PADS and DIS research evolves and expands to new domains, there is increasingly greater overlap between these two areas. For instance, synchronization and management of simulated time have become a growing concern in the DIS community. These issues are central to much of

Table 1: Contrasting PADS and DIS Research

	PADS	DIS
Speed Requirement	as-fast-as-possible	real-time
Typical Applications	VLSI circuits, telecomm, wargaming, transportation	military training, entertainment, air traffic control, emergency planning
Performance Metric	speedup	realism, scalability
Simulation Model	single model	federation of models
Distribution	single site	geographically distributed
Communication	arbitrary latency; reliable	100-300 milliseconds latency; unreliable
Network	multiprocessor or LAN	LAN and WAN

the work that has been done in the PADS community over the last twenty years. Similarly, there is an increasing emphasis on including constructive (war gaming) simulations in DIS; this is a domain where PADS algorithms have been used in the past, e.g., see Wieland, Hawley, Feinberg, DiLorento, Blume, Reiher, Beckman, Hontalas, Bellenot, and Jefferson (1989). Similarly, usage of PADS algorithms in real-time environments has received increased attention in recent years, e.g., see Ghosh, Panesar, Fujimoto, and Schwan (1994).

2 CONTRASTING PADS AND DIS

Table 1 summarizes some key technical differences between PADS and DIS research to date. One of the most important features distinguishing these communities is that DIS has been focussed primarily on real-time environments, while PADS work has focused on non-real-time "as fast as possible" simulations. This is because DIS has evolved from virtual training environments, while PADS has evolved from analytic simulation tools for engineering design.

In PADS, performance is paramount, and speedup relative to a sequential execution is used as the primary metric. By contrast, realism of the virtual environment is of principal importance in DIS, with *scalability* as a second, important goal. Intuitively, a distributed simulation is said to be scalable if it can be expanded to include more simulated entities executing on proportionately larger hardware configurations, and the simulator is still able to meet its stated objectives (e.g., value as a training mechanism), which in turn translates to real-time performance. While scalability is also important in PADS research, the issue has become a more pressing concern in DIS because of the military's desire to expand DIS demonstration exercises to include more simulated entities and sites.

A second distinction between PADS and DIS simulations concerns the models themselves. To date, PADS work has been largely concerned with execution of a single large simulation model. The components of the model are usually developed in a single simulation environment using one language, and are developed from the start as a single integrated system. Research has focused on simulation languages and tools to rapidly develop large simulation models. There is no question of interoperability among the different components of the simulation because they are designed to do so from the start. The question of "retrofitting" an existing model to execute on a parallel platform has received only a modest amount of attention, e.g., see Tsai and Fujimoto (1993) and Nicol and Heidelberger (1995). By contrast, achieving interoperability among existing and new simulation models is central to much of the work in the DIS community, and is one of the most difficult technical problems being attacked. The aggregate level simulation protocol (ALSP) project is a second project that bridges this gap by combining separately developed constructive simulators using a PADS synchronization protocol (Wilson and Weatherly 1994).

To date, most PADS research utilizes tightly coupled multiprocessors (using shared memory or message passing for communications) or LAN (local area network)-based distributed computing environments, while DIS exercises usually utilize LAN and WAN (wide area network) interconnects. Different assumptions are made by these communities concerning the network. PADS research generally assumes *reliable communications*, but *arbitrary communications latency*. This originates from the analytic nature of typical PADS applications. By contrast, most DIS work assumes unreliable communications but bounded maximum latencies for message delivery. Unreliable communications are used because message acknowledgment protocols that are used to ensure reliable delivery may compromise real-time performance. Message losses can often be tolerated because

many of the messages that are transmitted are simply periodic updates of state information, e.g., the position of vehicles, so subsequent messages allow the simulator to gracefully recover. Moreover, glitches in DIS simulations can often be tolerated, so long as they do not happen too frequently and do not have lasting effects. Maximum latencies also stem from the real-time nature of early DIS applications. Because of human limitations to perceive nearly simultaneous events, latencies up to 100 or 300 milliseconds are acceptable (DIS Steering Committee 1994); Correct modeling of more tightly coupled systems, e.g., components of a weapons system (say a simulated missile and its guidance system) require lower latencies (Cheung and Loper 1994).

The remainder of this paper is organized as follows. We first review the fundamental algorithms that have been developed in the PADS community for synchronizing as-fast-as-possible simulations. We then review the underlying design principles in DIS systems, and discuss and relate DIS techniques to research in the PADS community.

3 PADS RESEARCH

Much of the work in PADS is concerned with synchronization. The synchronization algorithm ensures that causally related events are processed in timestamp order. Toward this end, *conservative* and *optimistic* synchronization mechanisms have been devised. These mechanisms usually assume the simulation consists of a collection of *logical processes (LPs)* that communicate by exchanging timestamped messages or events. The goal of the synchronization mechanism is to ensure that each LP processes events in timestamp order; this requirement is referred to as the *local causality constraint.*

3.1 Conservative Synchronization

Historically, the first synchronization algorithms were based on conservative approaches. The principal task of any conservative protocol is to determine when it is "safe" to process an event, i.e., when have all events containing a smaller timestamp than the one being considered been received and processed by the LP. The algorithms described in Chandy and Misra (1979) assume the topology indicating which LPs send messages to which others is fixed and known prior to execution, and messages arrive on each incoming link in timestamp order. This guarantees that the timestamp of the last message received on a link is a lower bound on the timestamp of any subsequent message that will later arrive on that link.

Messages arriving on each incoming link are stored in first-in-first-out order, which is also timestamp order because of the above restriction. Each link has a clock that is equal to the timestamp of the message at the front of that link's queue if the queue contains a message, or the timestamp of the last received message if the queue is empty. The process repeatedly selects the link with the smallest clock and, if there is a message in that link's queue, processes it. If the selected queue is empty, the process blocks. This protocol guarantees that each process will only process events in non-decreasing timestamp order.

Although this approach avoids out-of-order event executions, it is prone to deadlock. A cycle of empty links with small link clock values (e.g., smaller than any unprocessed message in the simulator) can occur, resulting in each process waiting for the next process in the cycle. If there are relatively few unprocessed event messages compared to the number of links in the network, or if the unprocessed events become clustered in one portion of the network, deadlock may occur very frequently.

Null messages may be used to avoid deadlock. A null message with timestamp T_{null} sent from LP_A to LP_B is a promise by LP_A that it will not later send a message to LP_B carrying a timestamp smaller than T_{null}. Null messages do not correspond to any activity in the simulated system. Processes send null messages on each outgoing link after processing each event. A null message provides the receiver with additional information that may be used to determine that other events are safe to process.

How does a process determine the timestamps of the null messages it sends? The clock value of each *incoming* link provides a lower bound on the timestamp of the next event that will be removed from that link's buffer. When coupled with knowledge of the simulation performed by the process, this bound can be used to determine a lower bound on the timestamp of the next *outgoing* message on each output link. For example, if a queue server has a minimum service time of T, then the timestamp of any future departure event must be at least T units of simulated time larger than any arrival event that will be received in the future. Whenever a process finishes processing an event, it sends a null message on each of its output links indicating the best lower bound it can compute; the receiver of the null message can then compute new bounds on its outgoing links, send this information on to its neighbors, and so on. It can be shown that this algorithm avoids deadlock under some mild constraints (Chandy and Misra 1979).

This algorithm may generate many null messages, however. Another approach allows the computation

to deadlock, but then detects and breaks it (Chandy and Misra 1981). The deadlock can be broken by observing that the message(s) containing the smallest timestamp is (are) always safe to process. Alternatively, one may use a distributed computation to compute lower bound information (not unlike the distributed computation using null messages described above) to enlarge the set of safe messages.

Numerous variations on these approaches have been developed, as well as others. Some protocols use a synchronous execution where the computation cycles between (i) determining which events are "safe" to process, and (ii) processing those events. To determine which events are safe, the *distance between LPs* is sometimes used. This "distance" is the minimum amount of simulation time that must elapse for an event in one LP to directly or indirectly affect another LP, and can be used by an LP to determine bounds on the timestamp of future events it might receive from other LPs. Space does not permit full elaboration of all of the techniques that have been proposed, however, these techniques are reviewed in Fujimoto (1990) and Fujimoto and Nicol (1992).

3.2 Optimistic Synchronization

In contrast to conservative approaches that avoid violations of the local causality constraint, optimistic methods allow violations to occur, but are able to detect and recover from them. Optimistic approaches offer two important advantages over conservative techniques. First, they can exploit greater degrees of parallelism. If two events *might* affect each other, but the computations are such that they actually don't, optimistic mechanisms can process the events concurrently, while conservative methods must sequentialize execution. Second, conservative mechanism generally rely on application specific information (e.g., distance between objects) in order to determine which events are safe to process. While optimistic mechanisms can execute more efficiently if they exploit such information, they are less reliant on such information for correct execution. This allows the synchronization mechanism to be more transparent to the application program than conservative approaches, simplifying software development. On the other hand, optimistic methods may require more overhead computations than conservative approaches, leading to certain performance degradations.

The Time Warp mechanism (Jefferson 1985) is the most well known optimistic method. When an LP receives an event with timestamp smaller than one or more events it has already processed, it rolls back and reprocesses those events in timestamp order. Rolling back an event involves restoring the state of the LP to that which existed prior to processing the event (checkpoints are taken for this purpose), and "unsending" messages sent by the rolled back events. An elegant mechanism called anti-messages is provided to "unsend" messages.

An anti-message is a duplicate copy of a previously sent message. Whenever an anti-message and its matching (positive) message are both stored in the same queue, the two are deleted (annihilated). To "unsend" a message, a process need only send the corresponding anti-message. If the matching positive message has already been processed, the receiver process is rolled back, possibly producing additional anti-messages. Using this recursive procedure all effects of the erroneous message will eventually be erased.

Two problems remain to be solved before the above approach can be viewed as a viable synchronization mechanism. First, certain computations, e.g., I/O operations, cannot be rolled back. Second, the computation will continually consume more and more memory resources because a history (e.g., checkpoints) must be retained, even if no rollbacks occur; some mechanism is required to reclaim the memory used for this history information. Both problems are solved by *global virtual time (GVT)*. GVT is a lower bound on the timestamp of any future rollback. GVT is computed by observing that rollbacks are caused by messages arriving "in the past." Therefore, the smallest timestamp among unprocessed and partially processed messages gives a value for GVT. Once GVT has been computed, I/O operations occurring at simulated times older than GVT can be committed, and storage older than GVT (except one state vector for each LP) can be reclaimed.

Other optimistic algorithms have been proposed (Fujimoto 1990; Fujimoto and Nicol 1992). Most attempt to limit the amount of optimism. Typical techniques include using a sliding window of simulated time (Sokol, Briscoe, and Wieland 1988), or delaying message sends until it is guaranteed that the send will not be later rolled back, thereby eliminating the need for anti-messages, e.g., see Reynolds (1988).

3.3 Other Work

PADS research includes a substantial amount of work in other areas besides synchronization. Typical areas include massively parallel (so-called time-parallel) algorithms for specific simulation problems (e.g., simulation of communication networks), performance analysis, memory and workload management, simulation languages, hardware support, and applications (Fujimoto and Nicol 1992).

4 DIS RESEARCH

While the foundations for PADS research lies in early research concerning synchronization, the precursor to DIS was the SIMNET (SIMulator NETworking) project (1983-89) that demonstrated the viability of interconnecting several autonomous simulators in a distributed environment for training exercises (Kanarick 1991). SIMNET was used as the basis for the initial DIS protocols and standards, and many of the fundamental principles defined in SIMNET remain in DIS today. SIMNET realized over 250 networked simulators at 11 sites in 1990.

From a model execution standpoint, a DIS exercise can be viewed as a collection of autonomous simulators (e.g., tank simulators), each generating its own virtual environment representation of the battlefield from its own perspective. Each simulator sends messages, called *protocol data units (PDUs)*, whenever its state changes in a way that might affect another simulator. Typical PDUs include movement to a new location, firing at another simulated entity, etc.

In order to achieve interoperability among separately developed simulators, a set of evolving standards have been developed (IEEE 1278 1993). The standards specify the format and contents of PDUs exchanged between simulators as well as when PDUs should be sent.

DIS is based on several underlying design principles (DIS Steering Committee 1994):

- **Autonomy of Simulation Nodes.** Autonomy facilitates ease of development, integration of legacy simulators, and simulators joining or leaving the exercise while it is in progress. Each simulator advances simulation time according to a local real-time clock. Simulators are *not* required to determine which other simulators must receive PDUs; rather, PDUs are broadcast to all simulators and the receiver must determine those that are relevant to its own virtual environment.

- **Transmission of "Ground Truth" Information.** Each node sends absolute truth about the state of the entities it represents. Degradations of this information (e.g., due to environment or sensor limitations) are performed by the receiver.

- **Transmission of State Change Information Only.** To economize on communications, simulation nodes only transmit changes in behavior. If a vehicle continues to "do the same thing" (e.g., travel in a straight line with constant velocity), the rate at which state updates are transmitted is reduced. Simulators do transmit "keep alive" messages, e.g., every five seconds, so new simulators entering the exercise can include them in their virtual environment.

- **"Dead Reckoning" Algorithms.** All simulators use common algorithms to extrapolate the current state (position) of other entities between state updates. More will be said about this later.

- **Simulation Time Constraints.** Because humans cannot distinguish differences in time less than 100 milliseconds, a communication latency of up to this amount is required. Lower latencies are needed for other, non-training, simulators, e.g., testing of weapons systems.

We note that these design principles are seldom used in PADS research, but are pervasive in DIS work.

4.1 Dead-Reckoning

DIS simulations use a technique called *dead-reckoning* to reduce interprocessor communication to distribute position information. This reduction is realized by observing that rather than sending new position coordinates of moving entities at some predetermined frequency, processors can estimate the location of other entities through a local computation. In principal, one could duplicate a remote simulator in the local processor so that any dynamically changing state information is readily available. This local computation, when applied to computing position information of moving entities, if referred to as the *dead-reckoning model (DRM)*.

In practice, the DRM is only an approximation of the true simulator. An approximation is used because (1) the DRM does not receive inputs received by the actual simulator, e.g., a pilot using a a flight simulator decides to travel in a new direction, and (2) to economize on the amount of computation required to execute the DRM. In practice, the DRM is realized as a simplified, lower fidelity version is true model. To limit the amount of error between the true and DRM, the true simulator maintain its own copy of the DRM to determine when the divergence between them has become "too large," i.e., the difference between the true position and the dead-reckoned position exceeds some threshold. When this occurs, the true simulator transmits new, updated information (the true position) to "reset" the DRM. To avoid "jumps" in the display when the DRM is reset, simulators may realize the transition to the new position as a sequence of steps (Fishwick 1994).

4.2 Communications in DIS

Work in DIS is now attempting to scale exercises to include more entities and sites (locations). Current goals call for exercises including 50,000 entities at 30 sites by 1997, and 100,000 entities at 50 sites by 2000. Significant changes to DIS are required to enable simulations of this size, particularly with respect to the amount of communications that are required.

Even with dead-reckoning, the DIS protocol described above does not scale to such large simulations. An obvious problem is the reliance on broadcasts. There are two problems here: (1) realizing the communication bandwidth required to perform the broadcasts, estimated to be 375 Mbits per second per platform for a simulation with 100,000 players (Macedonia, Zyda, Pratt, Brutzman, and Barham 1995), is too costly, and (2) the computation load required to process incoming PDUs is excessive and wasteful, particularly as the size of the exercise increases because a smaller percentage of the incoming PDUs will be relevant to each simulator.

Several techniques have been developed to address this problem (Van Hook, Calvin, Newton, and Fusco 1994):

- **Relevance Filtering.** Rather than using broadcasts, messages are only sent to a subset of the simulation entities (Van Hook, Calvin, Newton, and Fusco 1994; Macedonia, Zyda, Pratt, Brutzman, and Barham 1995). For example, the battlefield can be divided into a grid, and entities need only send state update PDUs to entities in grid sectors in or "near" that generating the PDU. Relevance filters can be used for other information as well, e.g., radio communications.

- **Distributed Representation.** Information concerning remote entities can be stored locally. Caching is appropriate for information that seldom changes. Like dead-reckoning, remote computations can be replicated locally to generate dynamically changing data.

- **Compression.** Redundant information can be eliminated from the PDU. The *Protocol Independent Compression Algorithm (PICA)* uses a reference state PDU that is known to the communicating entities, and only transmits differences from the reference state (DiCaprio, Chiang, and Van Hook 1994). PICA has been report to yield four–fold compression of entity state PDUs (Van Hook, Calvin, Newton, and Fusco 1994).

- **Bundling.** Several PDUs may be bundled into larger messages to reduce overheads.

- **Overload Management.** These mechanisms reduce the communications load during periods of high utilization. For example, dead-reckoning thresholds may be adjusted to generate less traffic when the network is loaded.

- **Fidelity Management.** Different degrees of detail can be sent to different entities. For instance, less frequent state updates can be sent to distant receivers than those in close proximity. This is particularly useful for *wide area* receivers such as aircraft that can view large areas.

Relevance filtering and multicast communications go hand-in-hand, though it should be noted that relevance filtering still has merit even if multicast is not available. Multicast is more challenging in DIS than other applications (e.g., teleconferencing or video-on-demand) because of the need for a large number of multicast groups, and the dynamic nature of the groups. It is estimated that from 1,000 to 10,000 active multicast groups will be needed, with entities joining or leaving groups at a rate of hundreds per second. Changes to multicast groups should occur with low latency, e.g., one millisecond.

It is instructive to compare the approach used in DIS to distribute state information with the related problem of providing shared state variables between logical processes in PADS simulations. In PADS, a fundamental problem is that at any instant in real time, different logical processes will usually be at different points in simulated time, so care must be taken to ensure each LP receives the value corresponding to its current simulated time. Techniques using transactions-like protocols have been developed (Ghosh and Fujimoto 1991; Mehl and Hammes 1993). This problem is side-stepped in DIS because all simulators are nominally at the same current real-time of the simulation exercise. In practice, this is not the case because there will be variations in real time clocks at different simulator nodes, as discussed next.

4.3 Synchronization and Time Management

In DIS, *synchronization* usually refers to the problem of ensuring that the real-time clocks distributed throughout the network advance in synchrony with each other (Cheung and Loper 1994). *Time management* refers to the method used to advance simulated time in each simulator. The synchronization and time management mechanisms are responsible for ensuring *temporal correlation* is achieved, i.e., temporal aspects of the simulation exercise correspond to real-world behavior. While PADS simulation protocols guarantee that all logical processes observe the

same, timestamped ordered sequence of events, DIS makes no such guarantees. This is a well-known problem in DIS today. Correlation problems can occur because:

- Messages may be lost. DIS can tolerate some lost PDUs because many are simply updates of state information, and the simulator can simply wait for the next update to resynchronize itself with other simulators. Loss of other events, however, e.g., detonation of ordinances, could lead to more serious problems.

- No mechanism is provided to ensure that events are processed in timestamp order. PDUs may arrive out of order because of communication delay variations or differing (real-time) clocks in different simulators. Different simulators may perceive the same set of events in different orders, possibly resulting in different observed outcomes in different parts of the network. This is clearly undesirable.

In DIS, each PDU contains a timestamp with the *current* time of the simulator (obtained from the simulator's real-time clock) generating the PDU. This is in contrast to PADS simulations that typically generate events into the simulated future, i.e., with (simulated time) timestamp greater than the current time of the entity scheduling the event. Thus, events in DIS always arrive "late." Receivers can compensate by determining the communication delay in transmitting the message. *Relative timestamp* schemes do this based on past message transmission times, and *absolute timestamp* schemes assume synchronized real-time clocks in the sender and receiver to determine the latency by simply computing the difference between the send and receive times (Golner and Pollak 1994).

Much of the work in the DIS community to address the temporal correlation problem has been concerned with maintaining real-time clocks that are synchronized to a standard clock called *Coordinated Universal Time (UTC)*. Several approaches have been used for this task (Cheung and Loper 1994). Methods include broadcasting UTC on radio services, use of a U.S. National Institute of Standards and Technology (NIST) dial-up time service, use of a global positioning system (GPS) used by radio navigation systems (Kress, Phipps, and Carver 1994), and network protocols such as Network Time Protocol (NTP) to distribute clock information over the network (Mills 1992). The relationship between clock synchronization and temporal correlations is discussed in Katz (1994).

In addition to timing and synchronization errors, other correlation problems may arise due to difference in the virtual environments perceived by different simulator nodes. A tank that believes it is hiding behind a tree may actually be visible to other simulators because of differences in spatial computations. This can lead to "unfair" scenarios that reduce the realism of the exercise.

4.4 Other Work

Work in DIS encompasses a variety of other topics that are related to interoperability and producing realistic synthetic environments, as opposed to distributed execution. *Computer generated forces* use artificial intelligence techniques to generate automated or semi-automated models for forces, enabling the number of simulation participants to be much larger than the number of personnel participating in the exercise. *Aggregation and de-aggregation* algorithms enable interoperability between virtual simulators representing individual, de-aggregated entities (e.g., individual tanks) and constructive simulators with aggregated entities (a column of tanks) by aggregating and de-aggregating entities as needed. Work in *validation, verification, and accreditation* (VV&A) is concerned with defining appropriate performance metrics and measurement mechanisms to ascertain the extent that simulation exercises meet their goals. *Physical environment representation* is concerned with providing entities with common views of the battlefield in an environment changing because of man-made (e.g., introduction of craters when shells explode) and natural (e.g., roads washed out by thunderstorms) causes.

5 CONCLUSION

A substantial amount of effort has developed in two separate communities with the common goal of executing simulation programs on networked computing platforms. With expansion of DIS-related work into non-military applications, e.g., air traffic control, emergency planning, and entertainment, the impact of this work can be expected to increase in the years ahead. Although each community routinely uses different assumptions, goals, and even different vocabularies, we believe much can be gained by each in examining techniques used by the other. For example, synchronization methods developed by the PADS community may help to address temporal correlation problems now encountered in DIS. Similarly, techniques used to reduce communication in DIS might be applicable to some PADS simulation problems.

REFERENCES

Chandy, K. M. and J. Misra. 1979. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering SE-5*(5), 440–452.

Chandy, K. M. and J. Misra. 1981. Asynchronous distributed simulation via a sequence of parallel computations. *Communications of the ACM 24*(4), 198–205.

Cheung, S. and M. Loper. 1994. Synchronizing simulations in distributed interactive simulations. In *1994 Winter Simulation Conference Proceedings*, pp. 1316–1323.

DiCaprio, P. N., C. J. Chiang, and D. J. Van Hook. 1994. PICA performance in a lossy communications environment. In *11th Workshop on Standards for the Interoperability of Distributed Simulations*, Volume 2, pp. 363–366.

DIS Steering Committee. 1994. The DIS vision, a map to the future of distributed simulation. Technical Report IST-SP-94-01, Institute for Simulation and Training, Orlando, Florida.

Fishwick, P. A. 1994. *Simulation Model Design & Execution: Building Digital Worlds*. McGraw-Hill.

Fujimoto, R. M. 1990. Parallel discrete event simulation. *Communications of the ACM 33*(10), 30–53.

Fujimoto, R. M. and D. M. Nicol. 1992. State of the art in parallel simulation. In *1992 Winter Simulation Conference Proceedings*, pp. 122–127.

Ghosh, K. and R. M. Fujimoto. 1991. Parallel discrete event simulation using space-time memory. In *Proceedings of the 1991 International Conference on Parallel Processing*, Volume 3, pp. 201–208.

Ghosh, K., K. Panesar, R. M. Fujimoto, and K. Schwan. 1994. PORTS: A parallel, optimistic, real-time simulator. In 8^{th} *Workshop on Parallel and Distributed Simulation*, pp. 24–31.

Golner, M. and E. Pollak. 1994. The application of network time protocol (NTP) to implementing dis absolute timetamps. In *11th Workshop on Standards for the Interoperability of Distributed Simulations*, Volume 2, pp. 431–440.

IEEE 1278. 1993. Standard for information technology – protocols for distributed interactive simulation applications.

Jefferson, D. R. 1985. Virtual time. *ACM Transactions on Programming Languages and Systems 7*(3), 404–425.

Kanarick, C. 1991. A technical overview and history of the SIMNET project. In *Advances in Parallel and Distributed Simulation*, Volume 23, pp. 104–111.

Katz, A. 1994. Synchronization of networked simulators. In *11th Workshop on Standards for the Interoperability of Distributed Simulations*, Volume 2, pp. 81–87.

Kress, J., J. R. Phipps, and D. Carver, Jr. 1994. Synchronization of large scale distributed simulations and programs. In *10th Workshop on Standards for the Interoperability of Distributed Simulations*, Volume 2, pp. 611–623.

Macedonia, M., M. Zyda, D. Pratt, D. Brutzman, and P. Barham. 1995. Exploiting reality with multicast groups: A network architecture for large-scale virtual environments. In *1995 IEEE Virtual Reality Annual Symposium*, pp. 11–15.

Mehl, H. and S. Hammes. 1993. Shared variables in distributed simulation. In 7^{th} *Workshop on Parallel and Distributed Simulation*, Volume 23, pp. 68–75.

Mills, D. L. 1992. Network Time Protocol (version 3) specification, implementation, and analysis.

Nicol, D. M. and P. Heidelberger. 1995. On extending parallelism to serial simulators. In 9^{th} *Workshop on Parallel and Distributed Simulation*, pp. 60–67.

Reynolds, Jr., P. F. 1988. A spectrum of options for parallel simulation. In *1988 Winter Simulation Conference Proceedings*, pp. 325–332.

Sokol, L. M., D. P. Briscoe, and A. P. Wieland. 1988. MTW: a strategy for scheduling discrete simulation events for concurrent execution. In *Proceedings of the SCS Multiconference on Distributed Simulation*, Volume 19, pp. 34–42.

Tsai, J. J. and R. M. Fujimoto. 1993. Automatic parallelization of discrete event simulation programs. In *1993 Winter Simulation Conference Proceedings*, pp. 697–705.

Van Hook, D. J., J. O. Calvin, M. Newton, and D. Fusco. 1994. An approach to DIS scalability. In *11th Workshop on Standards for the Interoperability of Distributed Simulations*, Volume 2, pp. 347–356.

Wieland, F., L. Hawley, A. Feinberg, M. DiLorento, L. Blume, P. Reiher, B. Beckman, P. Hontalas, S. Bellenot, and D. R. Jefferson. 1989. Distributed combat simulation and Time Warp: The model and its performance. In *Proceedings of the SCS Multiconference on Distributed Simulation*, Volume 21, pp. 14–20.

Wilson, A. L. and R. M. Weatherly. 1994. The aggregate level simulation protocol: An evolving system. In *1994 Winter Simulation Conference Proceedings*, pp. 781–787.