Proceedings of the 1992 Winter Simulation Conference
ed. J. J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson

U.S. ARMY MODSIM ON JADE'S TIMEWARP

Dirk Baezner and Chuck Rohs

Jade Simulations International Corporation
Third Floor, 1422 Kensington Road N.W.,
Calgary, Alberta, Canada T2N 3P9

Harry Jones

U.S. Army TRADOC
ATRC-FPE, Building 193,
Ft. Leavenworth, Kansas

ABSTRACT

In May, 1991, Jade Simulations began work to develop an implementation of the U.S. Army's ModSim simulation language on Jade's TimeWarp. Incremental releases of that implementation were subsequently delivered to the U.S. Army, the most recent in June, 1992. This implementation represents the U.S. Army's second attempt to port ModSim to a Time Warp system. This implementation is the first involving Jade and Jade's TimeWarp.

In conjunction with Jade, the U.S. Army has begun porting a version of their Eagle combat simulation to the new implementation with encouraging preliminary results. Key modules of that simulation, collectively referred to as Core Eagle, have been ported and executed, with typical speedups between 6 and 10 times on 32 processors, compared with identical, sequential executions of the same simulation.

This paper summarizes the highlights of the reimplementation of ModSim and the Core Eagle performance study.

1 BACKGROUND

Jade's TimeWarp is a simulation executive for synchronizing distributed simulations. It is a commercial implementation of the Time Warp mechanism invented by Jefferson and Sowizral (Jefferson 1985) and is part of the Jade Simulation Environment (Baezner, Lomow, and Unger 1991), a C++-based development and run-time environment for creating and executing sequential and distributed simulations. The Jade Simulation Environment supports sequential and distributed simulations on networks of Sun/3 and Sun/4 workstations, and on the transputer-based Meiko Computing Surface. It is currently being ported to the HP-9000 700 series, and will be ported to the IBM RS-6000 series in the near future.

ModSim is a Modula-2-based object-oriented simulation language originally developed by CACI Products under contract to the U.S. Army (West 1985, West and Mullarney 1988). ModSim refers to the U.S. Army's version of ModSim, and not CACI's sequential simulation product, MODSIM II, although the two share a common ancestry and a largely similar syntax. From its Modula-2 heritage, ModSim inherits strong typing, modular decomposition, and much of its syntax. To this were added support for object-oriented programming and process-oriented simulation. The ModSim Environment includes a ModSim-to-C translator and a compilation manager that automates the compilation of projects.

The original development of ModSim included a prototype implementation of ModSim on the U.S. Army's Time Warp Operating System (TWOS) developed at JPL (Jefferson et al. 1987). The original design specified that simulation objects would inherit a *process* object and thereby become TWOS processes. Substantial difficulties in the implementation of that design resulted in an extreme simplification: each ModSim object was mapped directly to a TWOS process. To minimize the potentially significant message-passing overheads resulting from this simplification, applications needed to be developed such that only the simulation objects would be implemented as ModSim objects, whereas data aggregates would be implemented using records with associated procedures. Such an approach is consistent with other TWOS applications, written directly in C, that achieved significant speedup (Hontalas et al. 1989, Presley et al. 1989, Wieland et al. 1989).

The first ModSim application developers initiated their design based on the original distinction between objects and processes. Early in their design process, the implementation of ModSim was changed such that every object was a TWOS process. Unfortunately, the design of the application was not modified to reflect these changes. As a result, the implementation of that simulation was never able to run on TWOS (Rich and Michelsen 1991).

In 1990, Jade bid on and won the ModSim Support Contract to reimplement ModSim for use with Time Warp. The U.S. Army subsequently selected Jade's TimeWarp for the reimplementation of ModSim for several reasons:

1. Jade's TimeWarp is a commercial product that supports numerous users, projects, and simulation languages. This ensures a wide range of regular testing and evaluation that promotes a stable implementation. In addition, the U.S. Army automatically benefits from enhancements to Jade's TimeWarp funded by other projects.

2. Jade's TimeWarp includes direct support for ModSim's process view of simulation. This and other similarities between Jade's TimeWarp and ModSim reduced the time required to reimplement ModSim.

3. Jade's familiarity with Jade's TimeWarp implementation significantly reduced the learning curve involved in integrating ModSim with Time Warp, and eliminated the inevitable difficulties of integrating ModSim with a Time Warp implementation outside of Jade's control.

The simulation used to evaluate both the original implementation of ModSim on TWOS and the reimplementation of ModSim on Jade's TimeWarp is a version of the U.S. Army's Eagle combat simulation. Eagle was chosen by the U.S. Army because it is believed to be representative of the type of simulations for which ModSim will be used.

2 IMPLEMENTING MODSIM ON JADE'S TIMEWARP

The key lesson learned from the implementation of ModSim on TWOS was the need to create an appropriate fit between the paradigm of the language and the paradigm of the Time Warp implementation. Like TWOS, Jade's TimeWarp distinguishes between simulation objects, hereafter referred to as *entities*, and data aggregates, hereafter referred to as *objects*. Entities are represented by distributed processes, and objects are dynamically allocated data aggregates owned by whatever entity creates them. In addition, it is possible to create objects of global scope during the initialization of a simulation. These objects are subsequently accessible to all entities for read-only use. Global data was not supported in the implementation of ModSim on TWOS.

Given the current state of the art in distributed simulation, the entity/object paradigm is believed to be crucial for creating efficient distributed simulations. Specifically, it is important for programmers to identify which elements of a simulation should be represented by distributed processes and which elements should be represented by objects. It should be noted that the entity/object paradigm is not solely a distributed simulation paradigm. The paradigm originates with the sequential simulation language, Simula (Dahl, Myhrhaug, and Nygaard 1972), wherein simulation objects, called *processes*, are a special type of object capable of simulating delays and scheduling one another in simulation time.

Jade's review of the original implementation of ModSim on TWOS resulted in the recommendation that the entity/object paradigm be reintroduced into the language based on the conclusion that the paradigm provides the most direct mapping between the language and the Time Warp implementation (Baezner 1991). Following is a brief summary of the resulting paradigm. For a detailed description of the entire ModSim language, see (Jade 1992).

With the reintroduction of the entity/object paradigm, ModSim now supports both objects and entities. Objects are instances of object types and entities are instances of entity types. The two constructs are identical in syntax, except for the keywords **OBJECT** and **ENTITY** in the type declarations. For example,

```
EngineObj = OBJECT
    FuelCapacity : REAL;
    ...
    METHOD StartEngine();
    METHOD Refuel (IN amount : REAL);
END OBJECT;

AircraftEnt = ENTITY
    Engines : ARRAY [1 .. 2] OF EngineObj;
    ...
    METHOD TakeOff();
    METHOD Land();
END ENTITY;
```

As shown in the above example, **EngineObj** is an object type and **AircraftEnt** is an entity type. Both forms of type declaration support common object-oriented programming concepts, including single and multiple inheritance, dynamic binding, polymorphism, and private vs. public members. Currently, object types can only be derived from object types and entity types can only be derived from entity types.

Whenever a simulation creates an entity, it thereby creates an independently executing Time Warp process. Each entity can directly access its own fields and methods, as well as the fields and methods of any objects the entity creates. Since all such fields and methods are local to the entity, accesses to those fields and methods are implemented as C structure references and C function calls, respectively. In contrast, an entity interacts with other entities using **TELL** and **WAIT FOR** calls. For example,

```
aircraft : AircraftEnt;
...
CREATE(aircraft);
TELL aircraft TO TakeOff() IN 10.0;
TELL aircraft TO Land() IN 30.0;
```

In the above example, **aircraft** is declared to be a reference (i.e., a pointer) to an aircraft entity. The call to **CREATE** creates an aircraft entity and stores a reference to that entity in **aircraft**. The calls to **TELL** schedule the entity's **TakeOff** and **Land** methods to begin executing in 10.0 and 30.0 simulation time units, respectively. Unlike ordinary procedure calls, a **TELL** call does not wait for the called method to complete.

Alternatively, entities can wait for each other's methods to complete using **WAIT FOR** calls. For example,

```
WAIT FOR aircraft TO Land() IN 30.0;
...
```

will wait for the scheduled method to return before executing the next statement following the **WAIT FOR** call. In the case of **WAIT FOR**, the caller's simulation time is synchronized with that of the called method. For example, if we assume that **Land** executes in 20.0 simulation time units, the simulation time of the caller will have advanced by 50.0 simulation time units when the called method returns. This includes the time spent waiting for the called method to begin (30.0) plus the time spent waiting for the called method to execute and return (20.0).

Conceptually, each method invoked via **TELL** or **WAIT FOR** executes concurrently with all similarly invoked methods within the same entity. Each such method is referred to as an activity. In reality, the underlying entity implementation automatically interleaves the execution of an entity's activities in order of increasing simulation time. This form of concurrency is referred to as modeling concurrency and is distinct from the true concurrency achieved by Time Warp by simultaneously executing entities on different processors.

The implementation of a typical **TELL** call requires a single Time Warp message from the calling entity to the entity specified in the **TELL** call. A typical **WAIT FOR** call requires an additional message back to the calling entity so that the called method can return any **OUT** or **INOUT** parameters to the caller as well as signal the caller to proceed. In cases where the total size of all parameters in either direction exceeds the maximum size of a Time Warp message, the underlying implementation automatically separates the parameters into multiple messages, and subsequently recombines them in the receiving entity.

Due to strong similarities between ModSim and Jade's *Sim++* simulation library, the former is implemented in terms of the latter. *Sim++* is a library of C++ classes and functions that provides support for entity creation, scheduling and receiving messages, multi-part messages, random number generation, statistics collection, queueing, execution tracing, console and file input and output, and error handling. Implementations of *Sim++* exist for both sequential and distributed execution. The use of *Sim++* to implement ModSim eliminated the need to reimplement all of the above facilities directly on Jade's TimeWarp specifically for ModSim. In addition, since *Sim++* provides the same interface for all run-time environments (sequential and distributed, all architectures and operating systems), the implementation of ModSim is easily ported to any run-time environment supported by Jade.

3 CORE EAGLE

Because of differences in the paradigms of the two versions of ModSim, the original implementation of Eagle for use with ModSim on TWOS cannot be executed as is on Jade's TimeWarp. However, core modules from that implementation have been converted for use with the new paradigm. Collectively, these modules are referred to as Core Eagle.

Core Eagle is a time-stepped, ground combat simulation consisting of opposing red and blue forces fighting for control of a given terrain. Each side consists of one or more military units that scan and move over the terrain and engage enemy units.

Key attributes of military units include strength, speed, combat effectiveness, retreat threshold, and sensing range. Strength is the current strength of a military unit expressed in terms of tank equivalents. Speed is the maximum speed of a military unit. Combat effectiveness determines how quickly the strength of a military unit decreases in combat. When a military unit's strength falls below its retreat threshold, the military unit adopts a retreating posture. When a military unit's strength falls to zero, the military unit is considered destroyed and is removed from the simulation. The sensing range defines a circular area surrounding a military unit within which it can detect enemy units.

The terrain is divided into sectors, each of which is represented by an entity. Each military unit is represented by an object contained within whatever sector the military unit is currently located. As military units change their location, or move from one sector to another, sector entities inform the appropriate neighboring sectors, passing along military units and sensing information as required. Sector entities are also responsible for updating military unit strengths and postures as military units encounter enemy units in combat.

Military units can assume one of three postures: seeking, pursuing, or retreating. A military unit in a seeking posture moves over the terrain at random, scanning for enemy units to engage. When an enemy unit is detected, the seeking unit assumes a pursuing posture. A military unit in a pursuing posture engages all enemy units within sensing range while simultaneously attempting to overtake the nearest such unit. If all enemy units move out of a pursuing unit's sensing range, the pursuing unit returns to a seeking posture. A military unit automatically assumes a retreating posture when its strength falls below its retreat threshold. A military unit in a retreating posture attempts to avoid enemy units on the terrain. Once a military unit assumes a retreating posture, it never again assumes any other posture.

Military units typically move over the terrain along a calculated route of mobility corridors connecting a given origin with a given destination. A mobility corridor is any area of the terrain through which

movement is possible. Typically, mobility corridors form a fully-connected graph on the terrain. To represent different types of terrain, each mobility corridor has an associated mobility factor. The mobility factor is the percentage of a military unit's maximum speed at which the military unit is able to move through the corresponding mobility corridor.

Routes are calculated by the originating sector based on global mobility corridor data. The global data means that there is no need for sectors to communicate with one another to calculate a route, even if the route originates in one sector and terminates in another. However, this also means that the entire route is calculated in the originating sector, with no provision for distributing the calculation among multiple entities. Currently, the route calculation modules simply calculate *a* route, rather than the shortest route or the fastest route.

4 DISTRIBUTED DESIGN ISSUES

Three key design issues have been addressed by the Core Eagle implementation to maximize performance in the distributed environment: maximize the flexibility of sectors, minimize entity interactions, and minimize entity state sizes.

First, the number and size of sectors can be arbitrary. There can be as little as one sector that manages the entire terrain, or as many sectors as desired, each of which manages only a portion of the terrain. With fewer sectors, there is less potential for parallelism since there are fewer entities, but there is also less need for interaction among sectors, since the sectors are large, with fewer sector boundaries across which information must flow. The optimal number of sectors to use depends on other input parameters, such as the number of military units, their speed, their sensing range, and so on. However, as long as these parameters do not vary significantly from one run to the next, a single configuration of sectors was found to satisfy most runs.

Second, entity interactions were minimized to allow sectors to execute as independently as possible. Most notably, sectors use a start/stop protocol with neighboring sectors to moderate the flow of sensing information about enemy units. Once a sector has been told to start sending sensing information to a neighboring sector, it does so regularly until told to stop. In this way, a sector need not explicitly ask its neighbors for sensing information each and every timestep. The sensing information from neighboring sectors is required to handle cases where portions of the sensing range of a military unit coincide with portions of the terrain in neighboring sectors.

Third, entity state sizes were minimized to reduce the run-time cost of the state saving and rollback operations used by Jade's TimeWarp to synchronize a distributed simulation. Minimizing state sizes was accomplished primarily by maintaining static data in global memory. In particular, this includes the mobility corridors along which military units travel while crossing the terrain. In this way, a given military unit's route is reduced to a simple linked list of pointers to globally stored mobility corridors.

5 PERFORMANCE STUDY

All experiments were executed in parallel on Jade's TimeWarp on 12, 24, 32, 40, and 48 Computing Surface nodes with 8 megabytes of memory per node for the 12 node runs and 4 megabytes of memory per node for all other runs. Approximately 2.5 megabytes of memory per node is required by the Computing Surface operating system, the executable object code, and globally allocated data, leaving as little as 1.5 megabytes for entities, state saving, and message passing.

Each of the speedup graphs that follows is presented in terms of actual speedup. Actual speedup is defined as the total time to execute all entities sequentially divided by the total time to execute all entities in parallel. The time taken to execute all entities sequentially is calculated from runs with 36 sectors. For the input configurations used in this study, 36 sectors typically produced the lowest sequential execution times. The exceptions to this are the results for runs with a military unit density greater than 0.05 units per square km. The sequential time for these runs benefited from additional sectors and was calculated from runs using 100 sectors.

In addition to the actual speedup achieved by Jade's TimeWarp, corresponding graphs of the inherent parallelism of the model are given for 12, 24, 32, 40, and 48 processors. Although the speedup for any problem executing in parallel is limited by the number of processors on which it is run, real problems are further limited by causal relationships between entities and communication delays. Inherent parallelism is calculated by a performance analysis tool based on data collected by Jade's TimeWarp during the execution of the actual simulation. The tool simulates the execution of the same simulation using global knowledge provided by the data to ensure that each entity executes its messages in the exact order that it would in a sequential run, but on the same number of processors as the actual run. With its global knowledge, the tool eliminates the overhead imposed by the distributed synchronization mechanism. The resulting inherent parallelism is an upper bound on speedup for a given application on a given number of processors.

All performance results are compared to the following basic configuration of the Core Eagle model. The parameters for this configuration are believed to represent a typical scenario.

Units per square km	0.05
Sectors per dimension	10
Sensing range	3 km
Military unit speed	3 km/h
Timestep	6 minutes

All configurations of Core Eagle were run for 50 simulated hours. The number of sectors per dimension indicates the number of sectors from east to west and from north to south (e.g., 10 sectors per dimension would result in a simulation with 100 sectors). For the results reported here, the terrain was 100 km by 100 km and consisted of uniformly distributed mobility corridors, with each mobility corridor having a mobility factor of one. A uniform terrain was used to avoid the effects of bottlenecks possible with a non-uniform distribution of mobility corridors and mobility factors. Additional runs using a non-uniform terrain supplied by the U.S. Army typically reduced actual speedup by 5% to 15%, depending on the scenario.

Each of the parameters in the basic scenario was varied individually so that its effect could be analyzed separately. The list of values chosen for these parameters is given below.

Units per square km	0.02, 0.05, 0.1, 0.2
Sectors per dimension	6, 8, 10, 12
Sensing range	3, 5, 10 km
Military unit speed	3, 5, 10 km/h
Timestep	3, 6, 12, 24 minutes

For the sake of brevity, only the graphs for the basic scenario, the worst scenario, and the best scenario are shown. A more detailed presentation can be found in (Rohs and Baezner 1992).

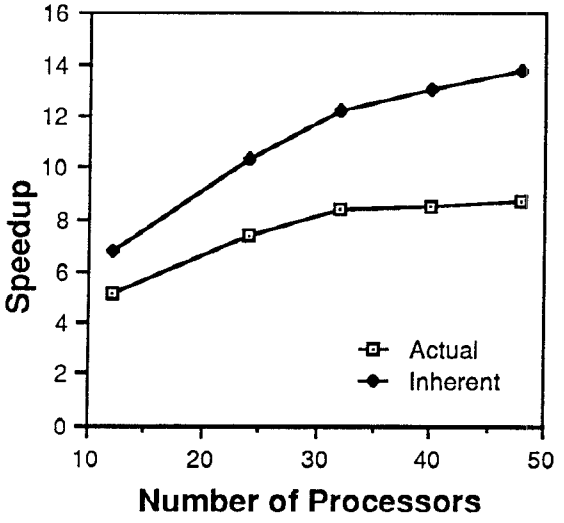

Figure 1 Actual speedup and inherent parallelism for the basic scenario.

Figure 1 shows the actual speedup and inherent parallelism for the basic scenario. The results for the basic scenario are typical of those achieved for most of the scenarios tested.

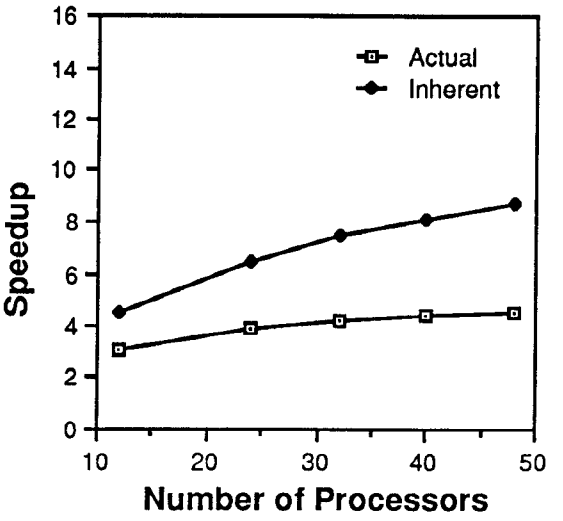

Figure 2 Actual speedup and inherent parallelism with sensing range = 10 km.

Figure 2 shows the actual speedup and inherent parallelism when sensing range is increased from 3 km to 10 km. This scenario resulted in the lowest speedup due to an increase in the number of sensing messages between neighboring sectors.

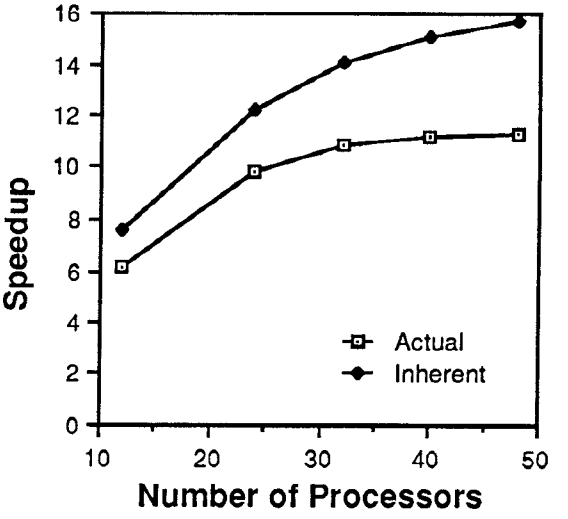

Figure 3 Actual speedup and inherent parallelism with units per square km = 0.2.

Figure 3 shows the actual speedup and inherent parallelism when the number of military units is increased from 0.05 units per square km to 0.2 units per square km. This scenario resulted in the highest speedup due to the increase in computation from additional military units.

As seen in these graphs, Jade's TimeWarp achieves between 70% and 80% of the inherent parallelism on 12 processors. The remaining 20% to 30% is the overhead for distributed synchronization. Due to memory constraints on 24, 32, 40, and 48 processors, the percentage of inherent parallelism achieved drops as low as 50% as the overhead for executing within limited memory grows. Nevertheless, this still allows for a near doubling of speedup between 12 and 48 processors. Less obvious from the graphs is that the inherent parallelism is typically only 30% to 60% of the number of processors. For example, on 48 processors, the basic scenario has an inherent limit of 14 times speedup, or 30%. Given the intended long-term use of Eagle, further effort toward improving the model's suitability for distributed execution is warranted.

6 ADDITIONAL RESULTS

In evaluating the performance of Core Eagle, it was found that attrition calculations were not as substantial in Core Eagle as in the original version of Eagle on which it is based. Furthermore, it was found that route calculations typically created momentary bottlenecks in the sectors engaged in calculating them. For example, route calculations of one second or more were observed in some sectors, while the average computation between timesteps was about 20 milliseconds. It is estimated that calculation of the shortest or fastest routes will require up to 30 seconds for a single route.

As a result of these findings, additional experiments were conducted in which the amount of computation between timesteps was artificially increased by an additional 10 milliseconds, and in which military units skipped route calculations and moved in straight lines.

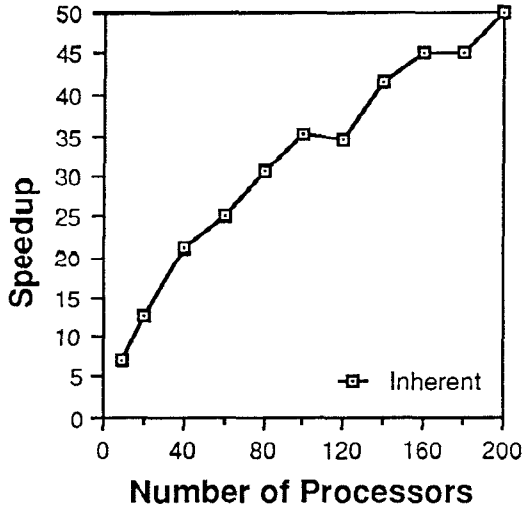

Figure 4 Inherent parallelism with added computation and no route calculations.

The increased computation is intended to reflect more substantial attrition calculations. The skipped route calculations are intended to reflect an implementation were the routes connecting any two mobility corridors can be either precomputed or computed *on the fly* as military units move from sector to sector. The extent to which this is possible is still being investigated.

Figure 4 shows the inherent parallelism up to 200 simulated processors when these assumptions are made for the basic scenario scaled up to 400 sectors. The maximum inherent parallelism is 65 when each of the 400 sectors is mapped to its own processor. Since the computation in this scenario is relatively uniform across all sectors, the peaks in the inherent parallelism curve occur when the number of sectors is an even multiple of the number of processors. Actual speedup in this case was 14.0 on the 48 processors actually available.

7 SUMMARY

The U.S. Army's ModSim simulation language has been successfully implemented on Jade's TimeWarp. The implementation includes support for all of ModSim's object-oriented programming constructs as well as a Simula-like, process-oriented modeling paradigm for sequential and distributed simulation.

Core modules of the U.S. Army's Eagle model have been successfully ported to the new implementation. Tests of Core Eagle show typical speedups of 6 to 10 times on 32 processors, compared to identical sequential executions of the same simulation. This represents as much as 70% to 80% of the inherent parallelism of the model as currently implemented. Additional assumptions concerning future changes to the model's implementation suggest that inherent parallelism may be as high as 50 on 200 processors or 65 on 400 processors.

Based on these results, the U.S. Army has extended the ModSim Support Contract until July, 1993. The additional year is expected to focus on further development of Core Eagle for use as a production quality combat analysis tool. This requires replacing existing Core Eagle modules with ones that implement more realistic attrition calculations, route calculations, and decision logic. Additional work to reduce route calculation bottlenecks are also required.

ACKNOWLEDGEMENTS

The authors thank Murray Peterson for his key role in the implementation of ModSim on Jade's TimeWarp, Alan Covington for his substantial assistance with the Core Eagle performance study, and the many Jade personnel and managers whose efforts and advice have contributed to the success of this project.

The authors also thank Brian Unger, the president of Jade, for reviewing this paper and suggesting additional improvements.

The implementation of ModSim on Jade's TimeWarp and the Core Eagle Performance Study were funded by the U.S. Army under ModSim Support Contract DABT60-90-D-0021.

REFERENCES

Baezner, D. 1991. *Preliminary Assessment of ModSim.* Jade Simulations.

Baezner, D., G. Lomow, and B. Unger. 1991. *Jade's Parallel Simulation Environment: Sim++ and TimeWarp.* Jade Simulations.

Dahl, O. J., B. Myhrhaug, and K. Nygaard. 1972. *Simula 67 Common Base Language.* Norwegian Computing Center Pub. S-52, Norwegian Computing Center, Oslo.

Hontalas, P., et al. 1989. Performance of Colliding Pucks Simulation on the Time Warp Operating System. *Proceedings of the SCS Multiconference on Distributed Simulation,* Simulation Series Vol. 21 No. 2, pages 3-7, The Society for Computer Simulation.

Jade Simulations. 1992. *ModSim: A Language for Object-Oriented Simulation,* Reference Manual, Release 2.1.

Jefferson, D. 1985. Virtual Time. *ACM Transactions on Programming Languages and Systems,* Vol. 7 No. 3, pages 404-425, Association for Computing Machinery.

Jefferson, D., et al. 1987. Distributed Simulation on the Time Warp Operating System. *Operating Systems Review (Proceedings of the Eleventh ACM Symposium on Operating System Principles),* Vol. 21 No. 5, pages 77-93.

Presley, M., et al. 1989. Benchmarking the Time Warp Operating System with a Computer Network Simulation, *Proceedings of the SCS Multiconference on Distributed Simulation,* Simulation Series Vol. 21 No. 2, pages 8-13, The Society for Computer Simulation.

Rich, D., and R. Michelsen. 1991. *An Assessment of the ModSim/TWOS Parallel Simulation Environment.* Los Alamos National Laboratory, Los Alamos, NM.

Rohs, C., and D. Baezner. 1992. *Performance Study of Core Eagle on Jade's TimeWarp.* Jade Simulations.

West, J. 1985. *Object-Oriented Distributed Simulation.* Technical Report, CACI.

West, J. and A. Mullarney. 1988. ModSim: A Language for Distributed Simulation. *Proceedings of the SCS Multiconference on Distributed Simulation,* Simulation Series Vol. 19 No. 3, pages 155-159, The Society for Computer Simulation.

Wieland, F., et al. 1989. Distributed Combat Simulation and Time Warp: The Model and its Performance. *Proceedings of the SCS Multiconference on Distributed Simulation,* Simulation Series Vol. 21 No. 2, pages 14-20, The Society for Computer Simulation.

AUTHOR BIOGRAPHIES

DIRK BAEZNER is a project manager at Jade Simulations where he led the development of ModSim on Jade's TimeWarp. Before that, he was the primary architect of much of the Jade Simulation Environment, including *Sim++*, the sequential run-time system, and a suite of performance analysis tools. Dirk Baezner has a B.Sc. and an M.Sc. in Computer Science from the University of Calgary in Canada. His Master's thesis was entitled *Language Design for Parallel Simulation.* He has since relocated to Ottawa to participate in the creation of a new Jade office focused on simulations of air traffic control systems and telecommunication networks.

CHUCK ROHS is a project manager at Jade Simulations where he now leads the development of Core Eagle and is simultaneously participating in the development of Jade's SS7-based telecommunication network model. Before that, he was responsible for much of the design and implementation of Jade's distributed VHDL effort. Before joining Jade, he participated in or led projects under contract to IBM, Honeywell, and NASA. He has a B.Sc. in Computer Science from the University of Calgary.

HARRY JONES is an Operations Research Analyst for the U.S. Army Training and Doctrine Analysis Command (TRAC) where he sponsored the development of a parallel processing capability for the ModSim language and has used this new version of ModSim to develop a prototype combat simulation. Before joining TRAC he worked for the U.S. Army Model Improvement Management Office where he managed the Simulation Technology Program. He has sponsored several projects to improve simulation technology, including the original ModSim development. Harry Jones has a B.S. in Physics from the University of Kansas and an M.S. in Operations Research from the University of Iowa.