Proceedings of the 1994 Winter Simulation Conference
ed. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila

DISTRIBUTED PARALLEL OBJECT-ORIENTED ENVIRONMENT FOR TRAFFIC SIMULATION (POETS)

Susan L. Mabry

Advanced Technology
& Design Center
Northrop Grumman Corporation
Pico Rivera, California

Jean-Luc Gaudiot

Department of Electrical
Engineering-Systems
University of Southern California
Los Angeles, California

ABSTRACT

POETS is an experimental traffic simulation incorporating parallel processing, object-oriented methods and data-flow scheduling on a distributed workstation platform. Primary emphasis of this project is on the integration of these advanced computing methods in the development of a fundamental traffic model. Traffic modeling provides an ideal application environment, representative of the demand for parallel processing in real-world simulations. Usage of distributed parallel processing, object oriented approaches, and data-flow scheduling in traffic simulation achieves objectives of enhanced performance, efficiency, representation and scalability in a pragmatic environment. The implementation employs C++ and a new parallel language system, Mentat. An overview of the project, challenges encountered, Mentat usage, and preliminary findings are presented in this paper.

1 INTRODUCTION

POETS depicts a simplistic transportation system in a mesoscopic simulation model incorporating approaches of parallel and distributed processing, object-oriented modeling and data-flow scheduling on a distributed workstation platform. The model is a hybrid system, possessing both continuous and discrete event properties.

Traffic modeling exemplifies the demand for parallel and distributed processing in real-world simulations. The potential quantity of vehicles and parallel events render sequential processing inadequate. It is our intention to explore and evaluate distributed and parallel processing techniques in improving efficiency and accuracy of large scale simulations. Parallel performance, accurate model representation, data distribution, application scalability and hardware scalability must all be realized at an effort and cost that make parallelism worthwhile.

Based upon extended C++(Stroustrup,1986), the Mentat Language System(Grimshaw,1993) combines features of the object-oriented paradigm with the data-driven computation model, described in Section 4.3. Mentat exploits a hybrid approach of explicit and compiler-based parallelism and interprocess communication management. Since Mentat uses the data-driven computation model, it is suited for message-passing, distributed memory architectures. Abstraction of complexity and encapsulation of behavior sustain integration, interconnection, and also a greater degree of parallel programmability.

This paper will begin with a discussion of motivation for this work and an introduction to the traffic model. Consideration is then given to the attractions and issues of parallel and distributed simulation, object-oriented approaches, data-driven computation model, and a brief overview of Mentat Language concepts. Implementation approaches and issues are examined. Finally, we shall discuss preliminary findings, further research and conclusions.

2 MOTIVATION

Traffic simulation is an important area for evaluation of traffic flow and traffic control. Simulation is being used to measure the effectiveness of technologies such as IVHS (Intelligent Vehicle Highway System). Intensive computation characteristics displayed in traffic simulation can benefit from parallel, distributed processing. Present methodologies in computer processing have become inadequate due to execution times for computation-intensive and data-intensive simulations. Interconnections of distributed systems and parallel processing of such distributed data is becoming a necessity for effective simulation, providing performance enhancement and more accurate event representation in real world systems. In the past, parallelism has often been achieved through brute force algorithms on expensive hardware platforms with limited scalability. This is an unacceptable application environment for widespread acceptance of parallelism because of the level of programming effort and cost effectiveness. It is our goal to explore approaches that provide more *user-friendly* development or *programmability* on an affordable distributed workstation platform.

Object-oriented methods are coveted for realistic data representation, encapsulation of costly communication overhead and as a basis for interconnection of models. Although the object-oriented paradigm offers advantageous features, these features are often obtained at the cost of performance and execution control. Data-driven scheduling offers a high degree of parallelism and execution upon principles of operand availability, thus significantly improving performance and execution control. Object-orientation addresses communication overhead associated with traditional fine-grain data-flow. Indeed, the blend of these two persuasions complement one another. We will explore the application of object oriented and data-flow methodologies within a C++ and Mentat programming framework.

Although this model is currently of limited scope with significant system boundaries in its initial state, the traffic flow type of model has been chosen because it will experience rapid growth of size and complexity. Consequently this model benefits tremendously from parallel and distributed processing. Clearly other simulations such as in manufacturing, medical or defense could similarly benefit from a *programmable* approach to parallelism and distribution.

3 MODEL DESIGN

POETS represents traffic flow of a freeway system as a mesoscopic simulation model. Every vehicle is viewed as a distinct entity and updated periodically while maintaining individual statistics for each vehicle object, thereby reflecting microscopic aspects. Yet the model also demonstrates macroscopic tendencies. Conditions or travel characteristics are determined by link status rather than individual vehicle trajectories. An individual vehicle object is advanced to another link when it has satisfied the required time to travel that link, the required time being a function of link attributes. Rate of travel on the link is derived from aggregate link conditions of capacity, volume, mean speeds, occurrence of incidents, day of the week and time of the day.

The network is viewed as having vehicle entity objects that travel through queue objects and are maintained by member functions which are also viewed as objects to be distributed for processing. Figure 1 depicts the interconnection nodes and classes representing objects in the traffic system. An inheritance hierarchy exists of lanes, ramps, links, sections, intersection nodes, freeways and regions. A vehicle is an entity possessing an origin and destination, local timer and allocated route. A lane is a queue of vehicles, with ramps being a specialized type of lane. Links are ordered collections of lanes with the same source and destination. Intersect nodes are connectors for links, consulting the router for future connections. Regions are collections of intersect nodes and their associated links. Regions provide scope of responsibility for routers and also assure validity of connecting links. For simplicity, we will hereafter refer to travel as in links, although advancement actually consists of travel through a hierarchy of lanes, links, ramps, sections, freeways and regions.

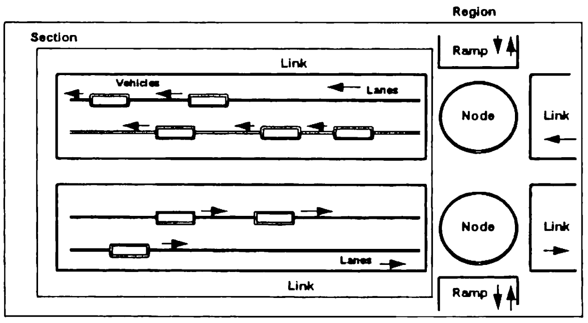

Figure 1: Logical Class Representation

The model displays properties of both continuous and discrete event simulation. At some levels the simulation varies continuously on a timed basis while on other levels the simulation responds to discrete events. For example, status of vehicles are updated in a continuous manner on timed intervals. However, the transitions from link to link, through intersection nodes, links, ramps and lane lists are in response to occurrence of events.

Provided to the executing program are command line arguments **time_to_run**, **day_of_week**, and **time_of_ day**. These arguments are used to establish initial base conditions of the regions and links. They are also used to establish the initial volume of vehicles and distribution intervals for introduction of additional vehicles during the simulation execution. Interval scanning is applied for the update of vehicle positions and traffic conditions. A central timing mechanism is provided for control of the overall simulation time. Individual timer objects are spun off with each vehicle object for the purpose of local vehicle advancement, such as within the lane object list.

Figure 2 illustrates the individual vehicle logic flow through the network. Vehicle objects require routing procedures to enter, exit or transition logically connecting regions, links or ramps. Routing structures include origin-destination matrices, route tables, route parsers and router. Advancement from links to nodes occur when a vehicle has satisfied the required time to travel that link. Routes or paths, are treated as linked lists to allow dynamic storage. Routing is currently performed on a static basis, routes are parsed and initially assigned to the vehicles. Our next stage of development incorporates dynamic routing assignments. Individual route assignment is maintained in the vehicle object with assignments currently made and verified on a node-by-node basis.

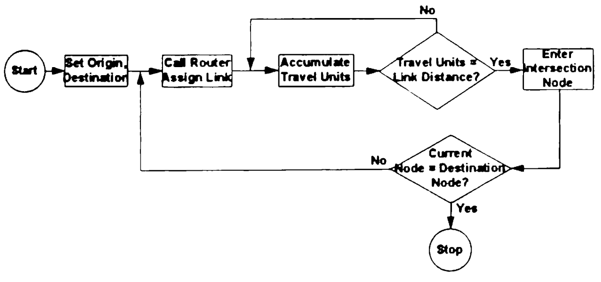

Figure 2: Logic of Individual Vehicle Movement

4 METHODOLOGIES

The goal of this experiment is to apply parallel and distributed processing, object-oriented paradigms, and data-driven scheduling to the design and implementation of this model. In doing so, we hope to explore implementation ease and methodology usefulness for general parallel simulations of real-world events. We will now briefly describe each of these concepts separately and related issues involved.

4.1 Parallel and Distributed Simulation

It is widely recognized that parallel and distributed processing provides solutions to current speed limitations of large network models, such as traffic simulations(Hsin and Wang,1992). Obvious objectives in parallel and distributed simulation are the exploitation of parallelism and the distribution of data and operations. These objectives are valued for qualities of faster performance, cost efficiency, and increased accuracy of model representation. However, these assets are not attained without difficulty.

Parallel and distributed processing under any type of application typically present scheduling, load balancing and communication overhead difficulties. The most significant difficulties facing distributed and parallel discrete event simulation efforts are synchronization and partition of events(Fujimoto,1990). The partitioned events must be distributed across a loosely coupled environment and processed concurrently for high performance, yet the events require timing and scheduling coordination. Furthermore, the approaches to address these challenges are what makes the environment less than hospitable to the user. The challenge for parallel simulation support is to provide efficient solutions to communication, load balancing, and scheduling issues, while taking full advantage of a distributed platform. A preferable environment would provide these solutions in a manner somewhat transparent to the user.

4.2 Object-Oriented Paradigm

Object-oriented approaches offer promise for the facilitation of dynamic linkage, efficient data structure handling, code reuse and expressive data representation (Booch,1993). The object-oriented paradigm is also inherently scalable. A disadvantage of object-oriented representation is the fact that performance frequently suffers. In our approach, data-driven scheduling and concurrent processing on multiple processors address this degradation. Data-driven scheduling, concurrent processing and encapsulation of costly communication overhead result in overall performance enhancement. Abstraction of complexity and encapsulation of behavior fundamentally support integration of such objects as vehicles in geographical objects and result in a greater degree of parallel programmability. Significant attention has recently been given to object-oriented paradigms. However, in order to truly reap the recognized benefits, alternate technologies such as data-driven scheduling must also be employed.

4.3 Data-Driven Computation Model

In the data-driven model(Gaudiot,1993), a program is considered a directed acyclic graph in which nodes represent instructions and the edges represent the data dependency relationship between the connected nodes. A data element produced by a node flows directly to its destination node(s) for consumption. An *actor* is then declared executable when its input data becomes ready for consumption. This means that every node is purely functional in that it causes no side-effect (data can only be read by those actors which need it). Also, sequencing the various parts of the computation is inherently parallel since the actors themselves can "decide" whether they are executable or not. This obliviates the need for a central controller such as the Program Counter in the von Neuman environment.

There are two different interpretation models of data-driven execution. The first is the *static* model pioneered by Dennis(Dennis,1991) in which only a single data token can exist on an edge. Although this simple model can be easily implemented, it is not flexible enough to handle execution mechanisms like recursion.

The other execution mechanism is the *dynamic* model pioneered independently by Arvind/Gostelow, 1982 and Watson/Gurd, 1982. This is a more flexible execution mechanism which allows multiple tokens to exist concurrently on an edge. In order to distinguish tokens that belong to different instances of a node, a tag field containing some unique identifier is added to a token. Thus a set of input tokens belonging to the same instance of a node can be identified through tag matching. While this model is more powerful than the static model, it is also more difficult to implement. Executable nodes as only fast as the token matching.

The data-driven computation model is recognized for its high degree of inherent parallelism. As explained in the following section data-driven concepts are incorporated into the Mentat Language System for the execution scheduling of distributed class objects.

4.4 Mentat Language System

Mentat is an object-oriented, parallel language system that has been developed at the University of Virginia under the leadership of Andrew Grimshaw. It combines features of the object-oriented paradigm with a hybrid data-driven computation model. Unlike traditional functional languages, state of objects may be designated for maintenance between executions thus preserving object-oriented persistence as necessary in some classes. Since Mentat uses a hybrid data-driven computation model it promises easy usability for previously difficult to program distributed memory MIMD architectures (Grimshaw,1990a).

The two facets of Mentat are the Mentat Programming Language(MPL) (Grimshaw and Liu,1988) and the Mentat Run-Time System(RTS) (Grimshaw,1990a). These dual components provide a combined approach of both explicit and compiler-based parallelism management. Mentat emphasizes the underlying assumption that the programmer is capable of making better granularity and partitioning decisions while the compiler more appropriately manages synchronization and communication. In Mentat, the programmer may thus designate partitioning based on familiarity with the problem domain. Interprocess communication and management of extensive and intricate asynchronous tasks are more easily handled by the compiler. The Mentat Run-Time System manages all aspects of communication, synchronization and scheduling(Grimshaw,1993b). Mentat developers stress that the programmer is thus freed to concentrate on domain and algorithms at hand rather than being concerned with communication primitives, scheduling, and load balancing.

The Mentat Programming Language consists of extensions to C++ with the objective of extending encapsulation principles to include encapsulation of parallelism. This extended encapsulation exploits intra-object as well as inter-object parallelism so as to claim an easy-to-use parallelism for distributed systems. An object has state, behavior and identity; the structure and behavior of similar objects are defined in their common class (Booch,1993). Inherent data parallelism is upheld through object inheritance and common attributes of objects.

The data-driven computation model in Mentat consists of graph-based, data-driven, medium grain, asynchronous and self-synchronizing aspects of data-flow. Program graphs are dynamically constructed by the Mentat front end compiler at run-time by determined data dependencies, allowing dynamic function binding as required by the object-oriented paradigm. Performing in an asynchronous manner, processing upon operand availability, and having little or no side effects, it is appropriate for message passing and distributed memory architectures such as the distributed workstation cluster.

5 POETS PROJECT

Timing, partitioning, scheduling and message-passing between processors all contribute to an extremely difficult environment for the user. We would like to accomplish the preceding issues on a traffic flow simulation in a programmable, user-friendly manner. Towards this goal Mentat is explored as a vehicle for achieving distributed and parallel processing of a simulation application in an easy-to-use manner.

5.1 Implementation Environment

Load balancing, process synchronization and interprocess communication are performed automatically by the Mentat Run-Time System. A round robin placement algorithm and sixty percentile

CPU load threshold transfer policy were enacted for this experiment. Running on each host of an active Mentat network are an Instantiation Manager daemon and a Token Manager Unit daemon (Mentat,1993). These background daemons coordinate the execution of the application objects. Existing clusters of Sun/SGI workstations were used as the distributed platforms. Workstations are extremely attractive as industry application platforms for parallel processing because of their power, versatility and affordability. The underlying architecture permits the processing to proceed as though on a MIMD architecture.

Independent objects are communicated and distributed for processing to autonomous host processors through asynchronous message passing. This allows multiple types of objects to be processed independently and concurrently across multiple workstations. Once the objects have been set, process scheduling of functions is performed at the compiler level without intervention by the programmer.

5.2 Language Implementation

The model was implemented in C++ with adaptations to Mentat. Use of C++ allows a powerful and commonly accepted foundation. The Mentat compiler incorporates embedded Mentat Run-Time System(RTS) calls into C++. The RTS provides instantiation and scheduling of Mentat objects, data-flow program graph construction and management. The C++ compiler is then invoked to generate executable C code.

A general logic overview of the **main** function of the system is provided in Figure 3. In **main** the program sets up the network, instantiating classes, declaring variables and calculating initial volume of traffic flow. Vehicles are then generated, assigned start times, vehicle identifications, random origins and destinations. An initial route is determined and assigned to the individual vehicle. The vehicle is then injected into the system. A distribution interval is determined for introduction of additional vehicles during the simulation run. The simulation is run until the period that the simulation has executed is greater than an input **time_to_run**. **Region->update** instigates a whole updating, testing, and advancing procedure, cascading down through sections, links, ramps and lanes. Also during the loop, additional vehicles are gradually introduced as periods exceed calculated distribution intervals.

```
main(int argc, char *argv[]) {
  // SETUP
  // DECLARATIONS AND INSTANTIATIONS
  // DETERMINE DAY_LOAD, TIME_RANGE, AND
       TYPICAL INITIAL VOLUME
       startup.calc_volume(day_of_week, time_of_day);

  // GENERATE VEHICLES
  for(veh_ct=1; veh_ct<gen_veh; veh_ct++){
    vehicle->genVehicle(startup.getVeh_id(),
         startup.getOrigin(), startup.getDest());
    vehicle->setroute(router->getroute(*vehicle));
    region->inject(*vehicle);
  }
  // SET DISTRIBUTION INTERVAL
  interval=startup.getDist_Interval();
  timer.setSimTime();

 // RUN the simulation
 while(simulation_time < time_to_run){
    // UPDATE,ADVANCE VEH THROUGH LINKS
     region->update();
    // INTRODUCE ADDITIONAL VEHICLES
     if (gen_clock > interval){
       vehicle->genVehicle(startup.getVeh_id(),
             startup.getOrigin(), startup.getDest());
        vehicle->setroute(router->getroute(*vehicle));
        region->inject(*vehicle);
        }
timer.setSimTime();
simulation_time=(float) timer.getSimTime();
gen_clock++;
  } }
```

Figure 3: Pseudocode of Main Program Routine

Class designation is the crucial aspect of implementation. Some class functions require full C++ sequentiality while others could greatly enhance performance by taking advantage of Mentat data-flow classes. Mentat offers a *regular mentat class* that provides pure data-flow parallelism with single assignment and no side effects. This class offers the highest parallel performance however will not maintain persistence. The *persistent mentat class* provides limited data-flow parallelism as is possible within the member functions of the class yet also maintains a state of persistence. The *sequential mentat class* provides an even more limited concurrency while maintaining an ordered execution and persistence. For greatest performance, developers strive for maximal usage of data-driven scheduling. This is achieved by usage in descending order of *regular mentat class*, *persistent mentat class* and *sequential mentat class* to traditional C++ classes with no parallelism. Various class designations are determined by the particular class's necessity for persistence of state or sequentiality of operations. The full range of class types were used in the model because of varying degrees of parallelism and persistence found in model activities.

It is important to realize that classes and also member functions encapsulated within those classes are all considered objects eligible for distribution. This fact enables both inter-object and intra-object parallelism.

Figure 4 provides a portion of a regular mentat class declaration and the associated definition of an encapsulated mentat member function **updateTimes**. Notice the mentat **rtf** call. This call forwards derived values to other classes having data dependencies.

```
persistent mentat class Vehicle {
private:
    int id, begin, end;
    float linktime, lifetime, simtime;
    clock_t time0, createtime;
    Route *route; Router *router; Timer timer;
 public:
    Vehicle(int ident, int b, int e) : id(ident), begin(b),
      end(e), linktime(0), lifetime(0),time0(0),
      createtime(timer->getSimTime()) { };
    void destroy();
    int getid(){return(id);};
    void setroute(Route &r);
   Route* getroute(){return(route);};
   int getbegin(){return(begin);};
   void update();
   void setTime0();
   void updateTimes(clock_t, clock_t);
    .......
   };

// Associated mentat member function declaration:
void Vehicle::updateTimes(clock_t tm0, clock_t cr0){
   linktime = (float) (timer.getTime(tm0));
   lifetime = (float) (timer.getTime(cr0));
   rtf(0);
   }
```

Figure 4: Example of Persistent Mentat Class

The Mentat class member functions provide logical partitions for distribution of objects across multiple processors. Parallel event decomposition is explored from the class member function and data-driven scheduling perspective. The class object designation allows design partitioning into inherent behavioral parallelism. This decomposition scheme bears resemblance to the "natural partitioning" concept of Blomquist and Brown(1994), exploiting the inherent parallel nature of the physical process. Object-orientation is also used to address communication overhead. Fine-grain parallelism is encapsulated within the member function units of decomposition, thus resulting in medium granularity and thereby reducing the amount of message passing required between multiple processors.

5.3 Initial Findings

Combined approaches of careful class design and Mentat usage address a majority of the parallel and distributed simulation issues. Mentat does pose some language constraints. Many of these constraints result from the support of a truly distributed environment while others are due to the evolving nature of the language. For instance, the standard C++ friend and static variables are justifiably not supported by Mentat. As globally shared entities, friend and static variables would interfere with distributed memory concepts and instances of Mentat classes treated as independent objects. Yet unsupported C++ features in Mentat classes include overloaded operators, templates and virtual functions. These lacking features limited full C++ object-oriented capabilities and also caused the greatest implementation problems.

Difficulties are to be expected both in supporting the notion of loosely coupled message passing architectures and within the framework of a developing language. In spite of implementation challenges encountered, Mentat certainly boasts appealing approaches for distributed and parallel processing. Unique in the usage of Mentat is the capability to stipulate classes with varying degrees of parallelism. The various types of classes execute concurrently on different processors with varying degrees of parallelism. This strength enhances scalability in such an application taking full advantage of the MIMD architecture, and generally provides a more flexible environment for parallelism than traditional SIMD architectures. Perhaps the most attractive feature of Mentat is the RTS where all scheduling, load balancing, and interprocess communication are handled automatically. Support for multiple load balancing policies by the RTS provides additional flexibility and caters to diverse applications.

At the time of this analysis, testing was performed on a network of 5 Sun SparcStations with minimal vehicles generated. As evidenced in the following graph, the system displays extremely efficient load balancing. Notice that a minimal number of mentat objects are scheduled for the host processor. This allows for additional overhead of originating host processor node activities. A nearly even distribution of objects occurs on the remaining processors. Figure 5 illustrates load balance.

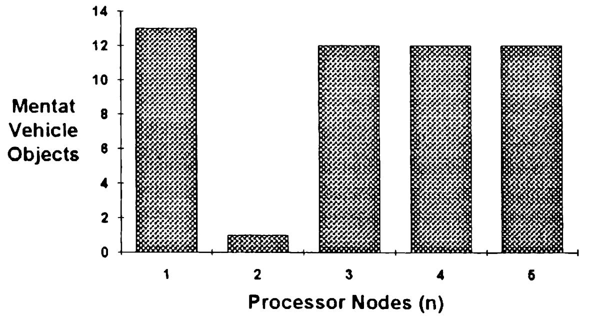

Figure 5: Load Distribution of 50 Mentat Objects

Speedup depicted in Figure 6 is consistent with expected trends of a small network with minimal objects. The speedup includes network communication costs a busy network. Speedup, S(n), is defined as the execution time on one processor over n processors for the given problem size of vehicles. Curves are low before and after peaking because of network interconnection overhead. These lows indicate when communication is as expensive as computational processing. The curves of 30 vehicles and of 50 vehicles begin to be indicative of the speedup of a larger scale model. Preliminary results are encouraging however inconclusive. Further testing is required on a larger network with greater number of objects.

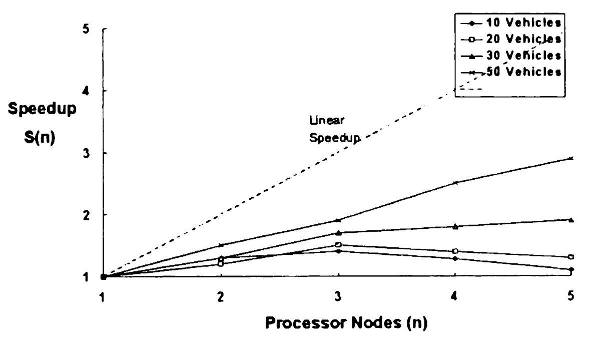

Figure 6: Speedup with Communication Costs

System efficiency, E(n), in Figure 7 reflects the relationship between speedup achieved and degree of processor node utilization. Minimum efficiency is equivalent to sequential execution. Maximum system efficiency is achieved when all processor nodes are fully utilized (Hwang,93).

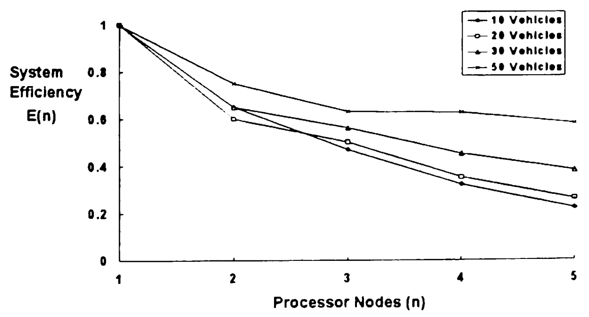

Figure 7: System Efficiency on Busy Network

6 Conclusions

This paper has presented a preliminary overview of an evolving model. The authors realize there remain several research issues that have not been fully addressed in this paper. In particular, we wish to explore dynamic routing algorithms for traffic flow. We also wish to enhance the model's origin-destination matrices and routing tables, addressing bottleneck issues encountered in model representation on a distributed network and frequent access to the routing structures. The object partitioning scheme and possible necessity of partial rollbacks require further investigation. In theory, data-driven scheduling precludes the necessity for rollbacks. Ongoing analysis of consistency and serializability of partitioned objects will either confirm or deny this hypothesis. Extensive performance analysis is required on larger networks with increased workload for conclusive results.

The project is distinctive in rather than being a theoretical problem, the traffic model is a realistic candidate for parallel processing with demanding circumstances. The application demonstrates significant parallelism and accurate event representation. Also being of sufficient complexity, the model provides an arena of typical problems encountered in normal implementation of large simulation systems. The project explored new methods to meet demands in a programmable manner. These employed methods are equally capable of supporting a wide range of applications.

In summary, although implementation has been quite challenging, initial findings have indicated promising results with the usage of Mentat. The RTS and Mentat class designations are especially beneficial towards the goal of achieving a usable parallel and distributed environment. As the Mentat Research Group augments support of C++ features in the compiler, the research language is anticipated to be increasingly applicable for practical applications. Further performance analysis is required on larger networks with increased workload for conclusive results. Utilizing object-oriented design in this approach, we have abstracted data and behavior for a more workable environment and encapsulated parallelism for minimization of communication overhead. Our approach of medium granularity with Mentat classes encapsulates and thereby minimizes communication overhead. Implicit data-driven scheduling affords parallelism, programmability and adaptability in the modeling environment.

REFERENCES

Arvind and Gostelow, K. The U-Interpreter, *IEEE Computer*, February, 1982.

Blomquist, R. N. and Brown, F. B. Parallel Monte Carlo Reactor Neutronics. *Proceedings of the Conference on High Performance Computing '94.* 1994. San Diego, California.

Booch B. *Object-Oriented Analysis and Design with Applications.* Benjamin/Cummings Publishing Company, Inc. 1993.

Dennis, J. The Evolution of Static Data-Flow Architecture. *Advanced Topics in Data-Flow Computing*, Prentice-Hall. edited by J-L Gaudiot and L. Bic. 1991.

Fujimoto, R. M. Parallel Discrete Event Simulation. *Communications of the ACM.* October 1990.

Gaudiot, J-L.. Data-Driven and Multithreaded Architectures for High-Performance Computing. *The International Summer School on Parallel Systems and Languages.* Praha, Czech Republic. July 1993.

Grimshaw, A. S. The Mentat Run-Time System: Support for medium grain parallel computation. *Fifth Distributed Memory Computing Conference.* Charleston, SC. April 1990

Grimshaw, A. S. . Easy-to-use object-oriented parallel processing with Mentat. Technical Report CS-92-32. University VA. May 1993

Grimshaw, A. S. and Liu, J. W. The Mentat programming language and architecture. *Workshop on Future Trends of Distributed Computing Systems.* Hong Kong. Sept 1988.

Hsin, V. J. K. and Wang, P. T. R. Modeling Concepts for Intelligent Vehicle Highway Systems Applications. *Proceedings of 1992 Winter Simulation Conference.* Washington, DC. 1992.

Hwang, K. *Advanced Computer Architecture.* McGraw-Hill, Inc. 1993.

Mentat Research Group. Mentat 2.6 Release Notes and Mentat 2.5 System Reference Manual. Dept. Computer Science, University of Virginia. 1993.

Stroustrup, B. *The C++ Programming Language.* Addison-Wesley, 1986.

Watson, I. and Gurd, J. A Practical Data-Flow Computer. *IEEE Computer*, February, 1982.

AUTHOR BIOGRAPHIES

SUSAN L. MABRY is a Software Engineer at Northrop Grumman Corporation Advanced Technology and Design Center. As a Northrop Grumman Fellow, she is also a Ph.D. student in Computer Systems Design Group, Information and Computer Science Department at the University of California, Irvine. She received both M.S. and B.S. Degrees in Computer Science from the University of Southern California in 1993 and the California State University Fullerton in 1991 respectively. Her research interests include parallel and distributed processing as applied to simulation, parallel architectures, data-flow scheduling, and distributed databases. She is a member of IEEE, ACM, and SCS.

JEAN-LUC GAUDIOT (S'76-M'82-SM'91) received the Diplome d'Ing'enieur from the Ecole Sup'erieure d'Ind'enieurs en Electrotechnique et Electronique, Paris, France, 1976, the M.Sc. and Ph.D. in Computer Science from the University of California, Los Angeles 1977 and 1982, respectively. Since 1982, he has been on the faculty of the Department of Electrical Engineering-Systems, University of Southern California, where is currently an Associate Professor. His research interests include data-flow architectures, fault-tolerant multiprocessors, and implementation of artificial neural networks. He has also consulted for several aerospace companies in Southern California. Dr. Gaudiot is a member of the ACM, the ACM SIGARCH, the IFIP Working Group 10.3 (Parallel Processing), and a senior member of the IEEE.