Proceedings of the 1989 Winter Simulation Conference
E.A. MacNair, K.J. Musselman, P. Heidelberger (eds.)

OLYMPUS: AN INTERACTIVE SIMULATION SYSTEM

Gary J. Nutt
Adam Beguelin
Isabelle Demeure
Stephen Elliott
Jeff McWhirter
Bruce Sanders
Department of Computer Science
Campus Box 430
University of Colorado
Boulder, CO 80309-0430

ABSTRACT

In this paper, we describe the Olympus Modeling System, a prototype modeling and simulation system we have built to interpret formal models of computation. Olympus employs a graphical, interactive user interface that enables one to control the system as it interprets a model. The system supports animation and simulation; animation is used to observe qualitative behavior of a model and simulation is used to obtain quantitative information about the model's behavior. The system supports multiple simultaneous users with fine-grained interaction between the users and the system.

1. INTRODUCTION

The *Olympus Modeling System* is an interactive, distributed model interpretation environment for bilogic precedence graphs (BPGs). BPGs are interpreted control flow graphs that incorporate conjunctive (AND) and disjunctive (exclusive OR) logic. Thus, very general control flow patterns (alternative, select, fork, and join) are supported by the model. BPGs also describe possible data flow among interpreted nodes. Like Petri nets, BPGs represent the status of the model through a distribution of tokens on nodes and edges. An interpreted BPG corresponds to a simulation model of some system.

An interactive model interpreter should provide several basic functions:

(1) It should have a mechanism for interactively creating and editing model instances.

(2) The user should be able to exercise a model with complete control over the interpretation, e.g., the user should be able to interrupt the interpretation at any moment (without setting breakpoints *a priori*).

(3) When an interpretation is interrupted, the user ought to be able to browse the state of the interpretation and even change the state prior to continuation.

(4) If the system is interpreting a model in scaled real time, then the user ought to be able to change the time scale while the model is in operation.

(5) The interactive system should also allow editing and interpretation to proceed in parallel, even though there will be times during which the user might leave the interpreter in an unusual state.

We address these requirements with Olympus by defining an underlying formal model for the simulation, then by implementing the user interface to the simulation interpreter as an asynchronous subsystem. This enables the user interface subsystem to respond immediately to user requests, even while the simulator is "busy" with other tasks. The two subsystems then communicate using conventional interprocess communication mechanisms.

In the remainder of the paper, we will first describe BPGs, then the design and implementation of the current Olympus frontend and backend. Finally, we will provide a simple example to illustrate the use of the system and the model.

1.1. Related Work

There are a number of interactive simulation systems used for performance prediction. In each case, there is a pictorial representation of the model of operation; the analyst uses graphical support tools to describe the model of operation in the particular language of representation. Commercial products are available to support graphical interfaces to simulation software [2, 5]. The representation is then used to define a simulation program of the model.

In some cases, the system focuses only on providing a graphical editor for constructing a machine-readable model; the model can then be translated into a traditional simulation program. SIMF is one example of this type of system; it provides an interactive editor for preparing SLAM programs [19].

In other cases, the system implements the formal model of computation as the basis of the simulation system, but does not provide a graphical user interface, as was done in our original Olympus implementation, e.g., see [16].

Newer systems incorporate a visual editor along with some form of machine to execute the resulting model (either a translator or an interpreter) under the control of the modeling system. The user specifies the model using the editor, then runs the simulation. Generally, the simulation can be invoked to run continuously, or single-stepped through event executions. Some systems allow the simulation to be halted so that the model or parameters can be changed, then the simulation can be restarted. The PAWS/GPSM simulation system [1,4,5], the Performance Analysis Workstation (PAW) [7], PARET [10], and Quinault [13] are all examples of this type of system.

GADD [9] is intended to simulate system modules that interact with other modules using messages. The focus of the model is on message traffic analysis, so all modules are simulated by corresponding simulation modules and the message traffic maps one-for-one with the target system message traffic (cf. Misra's discussion of distributed simulation [8]). As a consequence, it is possible to interconnect simulations with fully-implemented components, thus providing a testbed debugging environment.

2. THE BILOGIC PRECEDENCE GRAPH MODEL

Bilogic Precedence Graphs (BPGs) are composed from a set of *tasks*, a set of *control dependencies* among the tasks, and a specification of *data references* among tasks. A BPG is the union of a control flow subgraph and a data flow subgraph. The control flow subgraph consists of nodes that correspond to tasks and edges that specify precedence among the tasks. The node set for the data flow subgraph is the union of the task node set with another set of nodes representing data repositories; edges in the data flow graph indicate data references by the tasks.

The control flow subgraph is similar to the UCLA Graph Model of Behavior (GMB) [3,16] in that it specifies conjunctive ("AND") and disjunctive ("OR") input and output logic specifications for each task. Let small, open circles represent tasks with exclusive OR logic; and small, closed circles represent tasks with AND logic (see Figure 1). OR-tasks are enabled by control flow into any input arc, and upon task termination, control can flow out on any output arc. AND-tasks are not enabled until control flows to the task on every input arc, and upon termination control flows out every output arc. Large circles represent tasks with OR-input logic and AND-output logic; ordinarily, we only use single input and single output arcs on these circles since we use them to emphasize the notion of nontrivial processing.

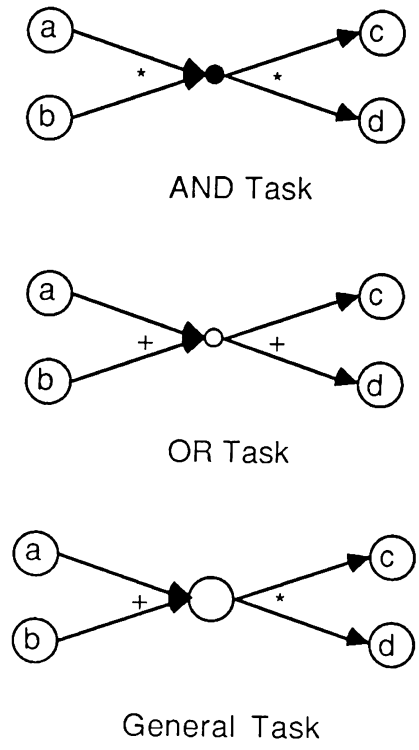


Figure 1: BPG Primitives

Following Petri nets, the control flow state is represented by tokens (data flow state is not represented explicitly in BPGs). Thus, one can think of markings and firings of the various tasks in the BPG just as in Petri nets. A BPG is activated by marking appropriate tasks with tokens, at which time the BPG firing rules (control flow logic rules)

describe sequences of markings corresponding to control flow among the tasks.

Data flow in a BPG is represented by adding data repository nodes to the control flow graph, and arcs interconnecting nodes and data repositories. Data repositories are meant to explicitly represent *possible* flow of data among individual tasks. The task interpretation will ultimately determine if data is read from (written to) a data repository by the task. The resulting model illustrates storage references and inter-task communication. Squares are used to represent data repositories in BPGs.

Figure 2 is a BPG of a simple system with two customers and one server. The server is represented by tasks s_1, s_2, and s_3. Tasks s_2 represents the case that the server is idle; at initialization, this task contains a token. Task s_3 is an AND-task which fires only when there is a token on arc (s_2, s_3) and another on arc (s_2, s_3). Whenever a token resides on task s_1, then the server is busy.

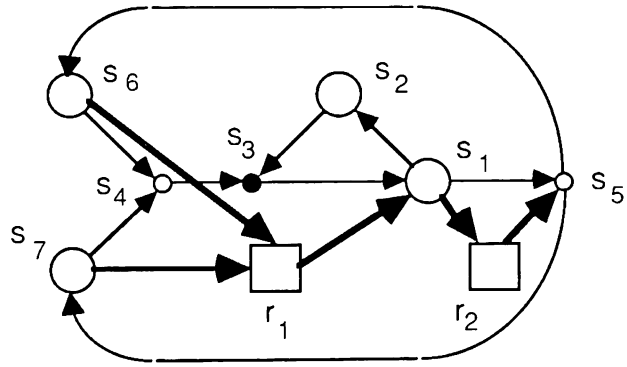


Figure 2: A 2-Customer Server

Tasks s_4 through s_7 model the customer requests for service. When a token is on s_6, then this represents the case when the first customer is "thinking" and does not require the service; s_7 represents a similar state for the second customer. Tasks s_4 and s_5 multiplex tokens (representing requests for service) into and out of the server.

In order to ensure that s_4 has enough information to demultiplex a token to the correct "thinking" task, it reads information from repository r_2 (placed there by s_1) to identify which customer was just serviced. Repository r_1 is used in a similar manner to provide the server busy task (s_1) with the corresponding information.

Each task may have a procedural *interpretation*, to specify the amount of time required for firing the task, and for performing miscellaneous simulation tasks. Thus, task s_1 can infer the desired amount of service time and manage the customer identity by evaluating a procedure similar to:

```
s1()
{
        struct *request;

        request = read_repository(r1);
        wait(request.service_time);
        write_repository(r2, request);
}
```

The interpretation is evaluated each time the task is fired. Tasks with OR (output) logic can use interpretations to specify deterministic behavior; the procedure evaluates information available to it (from repositories), then selects an output arc to receive the resulting token. For example, s_5 might look like:

```
s5()
{
        struct *request;

        request = read_repository(r2);
        if (request.customer == 'first')
                route(s6)
        else
                route(s7);
}
```

BPG tasks are hierarchical. Any task may be refined by defining a new BPG which has the same input/output behavior as the parent task. While this is an important aspect of BPGs (to address scaling problems) we do not discuss it further in this paper.

This is a brief, intuitive description of BPGs. Part of the motivation for using BPGs is that they are sufficiently simple, and similar to other commonly used models, that they are natural for representing the individual events involved in the simulation of a system. (A similar argument was used to justify the formulation of E-nets [11, 12]). The other rationale for using BPGs is that they encompass the semantics of several other formal models, including Petri nets, queueing networks, and several CASE models; by implementing the modeling system so that it interprets the semantics of BPGs, it is possible to provide a user interface that employs the syntax of these other models at a particular user's design workstation.

For the interested reader, a more complete and formal description of BPGs can be found in [15].

3. THE OLYMPUS MODELING SYSTEM

The BPG model provides a language for describing target system behavior, while Olympus provides a medium for expressing model instances, and for studying these models by observing their reaction to different conditions. By constructing an interactive system to support the model (using bit-map workstation technology), we have created an environment in which alternatives -- changes in loading conditions, changes in parameters, or changes in the model itself -- are easy to explore. Furthermore, the design decouples the user interface from the simulation itself, allowing the user to exercise highly interactive control over the simulation.

Olympus has been implemented in a network of Sun workstations, using Sun graphics and network software. We briefly describe the architecture and one of the implementations; additional details can be found in [14].

3.1. The Architecture

Olympus is an interactive system composed from a *frontend* and a *backend*, see Figure 3. The frontend implements the user interface, while the backend provides storage and interpretation of the model (independent of the frontend implementation).

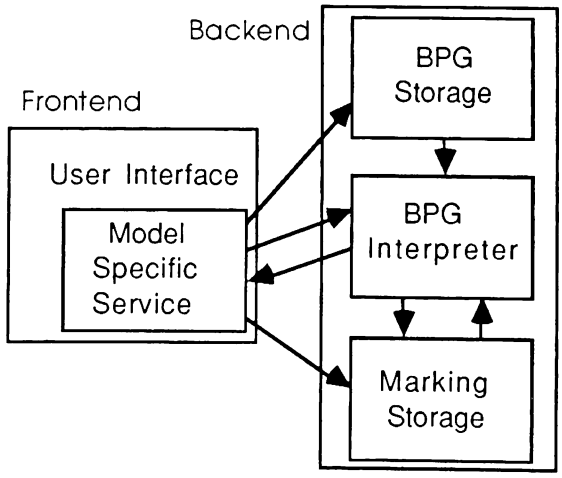


Figure 3: The Olympus Architecture

The Frontend

The separation of the system into a frontend and a backend allows the user interface to be independent of the simulation details. It also enables us to decouple the user interface console tasks from the simulation tasks, yet provide explicit means for the two parts to interact.

The frontend serves two main purposes: First, it implements the human factor (and many of the cognitive) aspects of the interaction between the user and the machine, i.e., it acts as a user interface. Second, it implements the syntax of the model or program specification, e.g., if the model of computation uses boxes to represent basic blocks of computation, then the frontend is responsible for drawing boxes, interconnecting them, etc.

For example, the frontend might take the approach that the user interface need only support a keyboard and a 25x80 character screen. In this case, the model or program is specified to the interface by typing some linear description of the model. The human factor aspects of the interface are issues such as keyboard mappings, escape characters for invoking commands, etc.

At the other end of the spectrum, the frontend may be based on a point-and-select graphics interface that provides a pallet of model primitives that can be placed on a "canvas," and that can be interconnected using arcs.

The frontend generates a structured internal representation of the model that is stored in the backend. In order to build consistent internal representations, the frontend performs syntactic analysis of the graph as it is being constructed by the user.

To the extent that the syntax of a particular model or language can be separated from the semantics, then the frontend can be made to be independent of the backend. Olympus provides a backend which implements the semantics of BPGs, so it is expected that the frontend could implement any of a number of models or languages of differing syntax that can be mapped into BPG semantics, see [15]. However, in this paper we only discuss a BPG frontend in use with the backend.

The Backend

The backend is an interpretation engine for BPGs, i.e., it executes the control flow of the graph, interpreting nodes as dictated by the BPG model. The backend reacts to directives from the frontend, then notifies the frontend of the changing model status as the interpretation process takes place. The backend is decomposed into parts to handle storage

of the model, storage of the marking, task interpretation, and repository interpretation.

Model and marking storage manage records representing atoms in the model, e.g., a task, and edge, or an interpretation for models and a token/task for markings. The task interpreter implements the BPG control flow semantics. It moves tokens around on the graph, evaluating interpretations as required. During evaluation, the task interpreter may invoke the repository interpreter as a function of the interpretation of a specific task. Thus, the repository interpreter acts as a server to the task interpreter client.

The frontend and backend communicate using a client-server protocol unique to Olympus. The protocol is based on asynchronous message passing primitives in which messages are formatted for efficient transformation of editing, interpretation, and control information between the two components. That is, the backend is a server which responds to commands from the frontend (a client). For example, when the user creates a new node at the frontend, the backend server is told to store the node; the server responds by updating the model storage and telling the frontend that there is a new typed object in the model. (The frontend can treat the object as it sees fit.)

The backend supports single-stepped interpretation of a model, as well as continuous operation. A single interpretation step refers to the occurrence of one BPG event, i.e., the initiation or termination of one task firing.

Continuous operation causes the interpreter to move tokens from task-to-task in real time, as determined from the task interpretations. In order to provide more flexibility for observing the dynamics of operation, the interpreter also provides a means by which the frontend can specify the ratio of real time to time used by the interpreter.

The backend will report summary interpretation information for each task and repository in the model. The report includes information about the activity of each task, expressed in terms of the number of times that the task was activated, and the amount of time that the task was active. Repository reports indicate the number of read and write references for the repository.

3.2. An Implementation

We have built several implementations of the frontend and backend; here we summarize only the most recent ones. Figure 4 illustrates an implementation of the architecture in a network of Sun workstations, using Unix (R) processes, graphics, and network protocols.

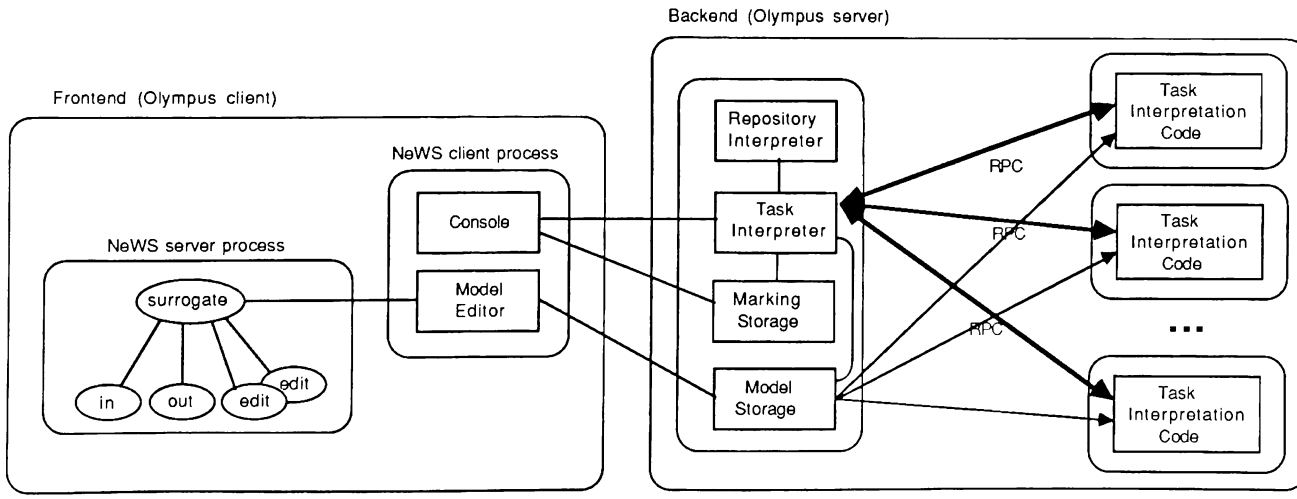


Figure 4: An Olympus Implementation

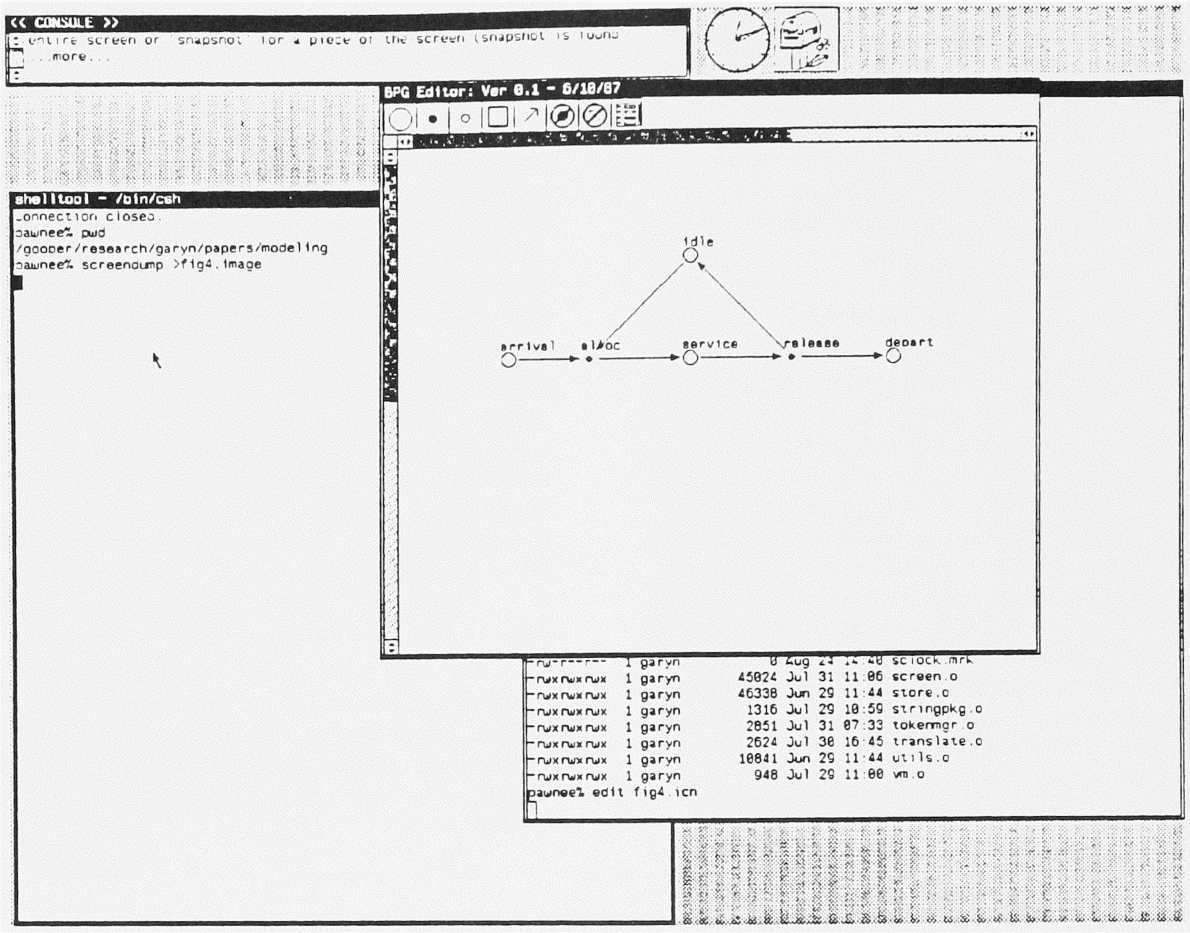


Figure 5: The SunView Olympus Frontend

There are several different versions of the frontend in use: The first point-and-select version is the most functionally complete, although not the most aesthetically pleasing. It was built on the SunView [20] window package as a single Unix process. Figure 5 is the display for this frontend. As an experiment, we also developed a version on a Symbolics Lisp workstation [17], however we have not continued to maintain this version.

The newest frontend is being implemented as two Unix processes that conform to Sun's NeWS model [22]. That is, the Olympus client process is actually implemented as a NeWS client and a NeWS server process. The NeWS client implements the logical aspects of the user interface, while the NeWS server process -- implemented as a community of lightweight processes called "in," "out," and "edit" in Figure 4 -- is responsible for placing images on the display. Thus, the model editor is the only part of the frontend that needs to interact with the model storage in the Olympus server. Also, NeWS allows its client and server processes to be in execution on distinct machines.

The Olympus server (backend) is implemented as n+1 Unix processes: The first process multiplexes among the four interpretation and storage "subprocesses." The other n processes are used to evaluate BPG interpretations.

BPG interpretations can be defined in any language, provided that the definition can be viewed as a procedure callable in C. The task interpreter uses the Sun Remote Procedure Call (RPC) facility [21] to invoke the interpretation procedure whenever the corresponding task is fired. In order to promulgate concurrency in the simulator, we have used the RPC facility as a remote fork rather than as a procedure call; thus, n tasks can be interpreted at one time by starting n RPC servers on different machines.

The frontend-backend interface is implemented on top of sockets [21]. This allows the frontend and backend to be executed on distinct machines in a network environment.

The separation of the frontend from the backend has allowed us to implement the interface so that there can be an arbitrary number of clients connected to the same server. Each client can send editing or console requests to the server, and the server will respond to all clients as if they were one, since they are connected to the server via a single, shared socket. This allows multiple users to view a single server session with full access and viewing rights. Since the server serializes each transaction, the users will not cause the server to violate critical sections or otherwise violate concurrency constraints.

4. AN EXAMPLE

Suppose that we were configuring an internet as a composition of four different networks. There are several different configurations that one might consider, e.g., see Figure 6. In Figure 6a, three gateway machines interconnect the four nets; two gateways are used in in Figure 6b, and a single centralized gateway is used to interconnect the machines. in Figure 6c.

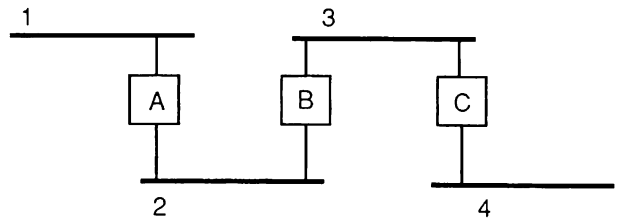
The gateway machines in the configuration for Figure 6a should not be as expensive as those in Figure 6b, and the single gateway in Figure 6c should be the most expensive (highest performance).

The single-gateway configuration may be the most cost-effective solution, particularly in cases where hosts on each subnet need to communicate an equal amount with all other hosts in the internet. However, a machine that is fast and large enough to support this configuration may be too costly. In this case, other configurations should be considered

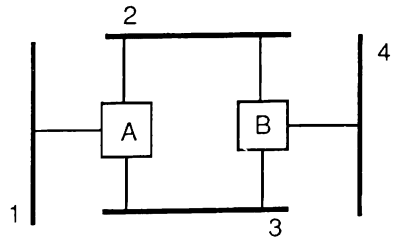
We will use this general scenario to describe how BPGs and Olympus can be used to quickly examine various scenarios (our discussion is far from complete, due to space limitations).

4.1. Modeling the Internet Configurations

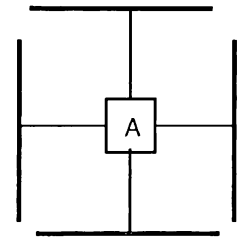
We can reuse part of the model shown in Figure 2 to represent a gateway machine. In this application it is not necessary to re-enable two customers, so tasks s_5 through s_7 are not needed. Consider the BPG shown in Figure 7: The service request population is now modeled by four submodels (g_1 through g_4) representing the four subnets.



6(a) Triple-Gateway Configuration



6(b) Double-Gateway Configuration



6(c) Single-Gateway Configuration

Figure 6: Possible Internet Configurations

Each subnet produces a token by firing, then reenables itself to produce a token at some later time. Each g_i has the form:

```
g_i()
{
        wait(sample(interarrival_distribution));
```

The service time of the gateway machine can be modeled by a constant distribution, or by something more complex if the gateway machine is more complex. (For example, the gateway may not be reliable if it is saturated, i.e., it may drop packets. Our simple model does not take this into account.)

The BPG shown in Figure 7 is easily constructed in Olympus, and can be viewed as an animation to gain a qualitative feel for the relative power required from the single gateway machine to satisfy various loads (as determined by the g_i distributions) As the gateway becomes saturated, tokens will build up on the arc from s_4 to s_3.

Olympus allows multiple users to view the animation, and allows the graph to be edited while the animation is in progress. For example, one could add a fifth network to the simulation model without halting the animator.

Figure 8 is a model of the configuration shown in Figure 6b. In this model, some fraction of the load from network 2 goes to gateway (server) A and the other portion goes to gateway B. The relative speeds of the two gateways can be decreased by increasing the mean of the service time distributions.

Figure 9 is a model of the configuration shown in Figure 6a. Again, the loads are split for networks 2 and 3, since they each have two gateway machines.

Olympus is used to create the models, to observe their behavior in qualitative terms, then to obtain quantitative performance data about their behavior. (The average number of tokens on the arcs incident into $s_{x,3}$ reflects the amount of packets that need to be buffered at a gateway.)

It is easy to explore a wide variety of gateway machine performance considerations by adjusting the service time distribution of the gateway server tasks. The performance is again reflected by the token dwell time at arcs that represent the gateway queues.

5. SUMMARY

Olympus is a working prototype modeling system. It is currently being used to model memory access strategies in MIMD machines [18], even though it continues to be developed.

The frontend/backend architecture has provided considerable implementation freedom for focusing on the simulation or on the user aspects of the system. It has also been the major factor in meeting the basic requirements described in the Introduction. Since the frontend is a separate process from the simulator, it is responsive to the commands and queries of the user. The backend has been designed to field messages from the frontend as part of it's basic simulation cycle, thus it, too, is responsive to the user control without undue complication. Because the frontend and the backend communicate using a socket pair, any number of frontends can connect to the socket at one time. The backend will send all information needed by a frontend to the socket, where each instance of the frontend will react to the information by updating a screen, moving a token, etc. Messages from the frontend to the backend will be serialized by the socket, thus each frontend can act as a console, i.e., backend transactions are atomic.

The BPG model provides a graph model of the computation describing a simulation program. While we have distributed the simulation program interpreter (task interpretations are executable on different machines), we are currently exploring techniques for distributing the model interpretation -- both by distributing the model and by further distributing simulation functions. One goal of this research is to address simulation strategies that lie between the conservative Chandy-Misra technique [8] and the optimistic time warp technique [6].

Our experience with Olympus and BPGs has shown us that the architecture enables us to implement systems that meet the goals described above. However, we continue to refine our language and our architecture. For example, newer languages will incorporate data on tokens, and will place more constraints on the form of the interpretations (currently, they are any C procedure). Our current version of Olympus provides limited support for hierarchical BPGs, which we believe is a fundamental requirement for scalable systems; we are currently designing new facilities to handle hierarchies more generally.

ACKNOWLEDGEMENTS

This research has been supported by NSF Cooperative Agreement DCR-8420944, NSF Grant No. CCR-8802283, and a grant from U S West Advanced Technologies.

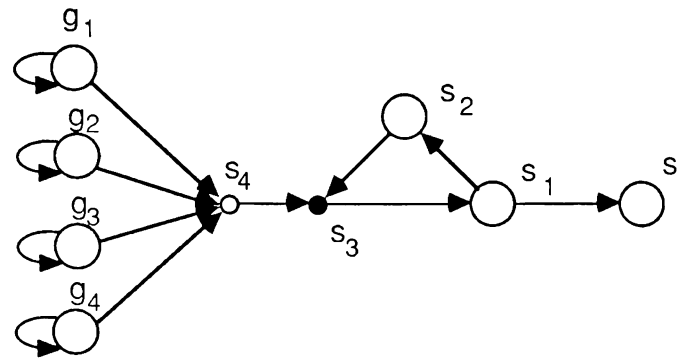


Figure 7: The Single-Gateway Model

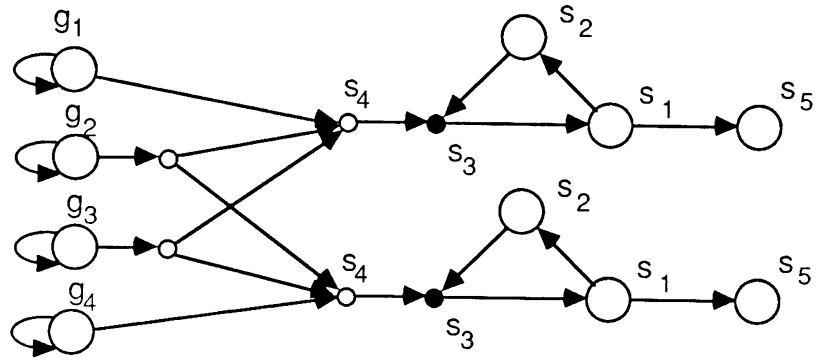


Figure 8: The Two-Gateway Configuration

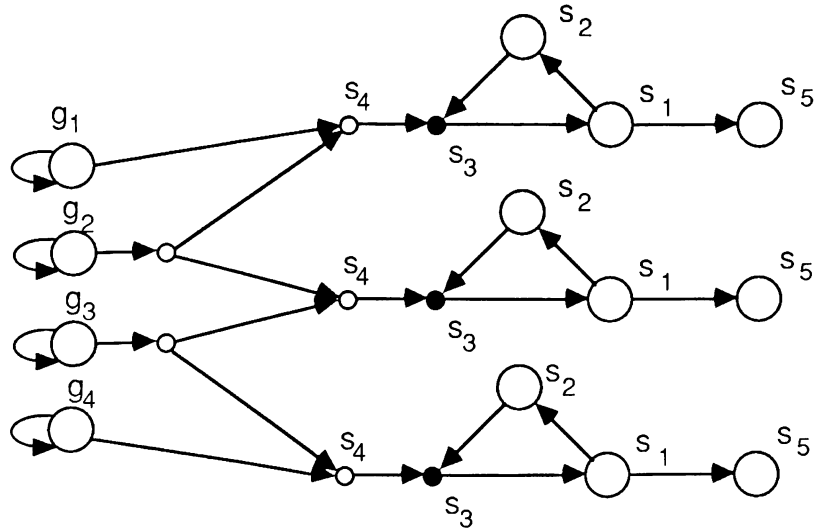


Figure 9: The Three-Gateway Configuration

6. REFERENCES

1. J. C. Browne, D. Neuse, J. Dutton and K. Yu, "Graphical Programming for Simulation of Computer Systems", *Proceedings of the 18th Annual Simulation Symposium*, 1985.
2. *CACI Simscript II.5 marketing information*, CACI Product Company, La Jolla, California, 1988.
3. G. Estrin, "A Methodology for Design of Digital Systems -- Supported by SARA at the Age of One", *AFIPS Conference Proceedings of the National Computer Conference 47* (1978), 313-324.
4. S. Iacobovici and C. Ng, "VLSI and System Performance Modeling", *IEEE Micro*, August 1987, 59-72.
5. *PAWS/GPSM marketing brochures*, Information Research Associates, Austin, TX, 1988.
6. D. Jefferson, B. Beckman, F. Wieland, L. Blume, M. DiLoreto, P. Hontalas, P. Laroche, K. Sturdevant, J. Tupman, V. Warren, J. Wedel and H. Younger, "Distributed Simulation and the Time Warp Operating System", *Proceedings of the Eleventh ACM Symposium on Operating Systems Principles*, Austin, Texas, November 1987, 77-93.
7. B. Melamed and R. J. T. Morris, "Visual Simulation: The Performance Analysis Workstation", *IEEE Computer 18*, 8 (August 1985), 87-94.
8. J. Misra, "Distributed-Discrete Event Simulation", *ACM Computing Surveys 18*, 1 (March 1986), 39-65.
9. M. Moser, "GADD -- A Tool for Graphical Animated Design and Debuggin", *ICC '87 Conference Record*, 1987, 38.2.1-38.2.5.
10. K. M. Nichols and J. T. Edmark, "Modeling Multicomputer Systems with PARET", *IEEE Computer 21*, 5 (May 1988), 39-48.
11. J. D. Noe and G. J. Nutt, "Macro E-Nets for Representing Parallel Systems", *IEEE Transactions on Computers C-12*, 8 (August 1973), 718-727.
12. G. J. Nutt, "The Formulation and Application of Evaluation Nets", Ph.D dissertation, Computer Science Group, University of Washington, 1972.
13. G. J. Nutt and P. A. Ricci, "Quinault: An Office Environment Simulator", *IEEE Computer 14*, 5 (May 1981), 41-57.
14. G. J. Nutt, "A Flexible, Distributed Simulation System", *Tenth International Conference on Application and Theory of Petri Nets*, Bonn, West Germany, June 1989.
15. G. J. Nutt, "A Formal Model for Interactive Simulation Systems", Technical Report No. CU-CS-410-88, Department of Computer Science - University of Colorado, Boulder, September 1988 (Revised May 1989).
16. R. R. Razouk, M. Vernon and G. Estrin, "Evaluation Methods in SARA -- The Graph Model Simulator", *Proceedings of the Conference on Simulation, Measurement and Modeling of Computer Systems*, August 1979, 189-206.
17. D. Redmiles, "Vesuvius Editor for Olympus (working title)", University of Colorado Department of Computer Science technical report (in preparation), 1988.
18. C. J. C. Schauble, "A Memory Access Simulator for MIMD Machines", University of Colorado, Department of Computer Science, PhD proposal, April 1989.
19. K. L. Stanwood, L. N. Waller and G. C. Marr, "System Iconic Modeling Facility", *Proceedings of the 1986 Winter Simulation Conference*, December 1986, 531-536.
20. "SunView Programmer's Guide", Document Number 800-1324-03, Sun Microsystems, Inc., February 1986.
21. "Networking on the Sun Workstation", Document Number 800-1345-10, Sun Microsystems, Inc., September 1986.
22. *NeWS: A Definitive Approach to Window Systems*, Sun Microsystems, Inc., 1987.

AUTHORS' BIOGRAPHIES

GARY J. NUTT is a Professor of Computer Science at the University of Colorado. He received the Ph.D in Computer Science from the University of Washington in 1972, and has held research positions at Xerox PARC and Bell Labs, and managment positions at NBI and Interactive Systems. His current research interests focus on modeling and performance for distributed systems.

ADAM BEGUELIN is a Ph.D. candidate in the Department of Computer Science at the University of Colorado. He has a B.S. degree with highest honors in Mathematics and Computer Science from Emory University in 1985 and an M.S. degree in Computer Science from the University of Colorado in 1988. His research interests include parallel processing, programming languages,

distributed systems, human computer interfaces, and graphics.

ISABELLE DEMEURE is a postdoctoral researcher in the Department of Computer Science at the University of Colorado. She received a Maîtrise de Mathématiques from University Paris VI, France, in 1981, a Diplôme d'Ingénieur from the Ecole Nationale Supérieure des Télécommunications de Paris, France, in 1983. She worked as a systems and networks software engineer at SESA, in France, from 1983 to 1985. She was granted the Ph.D degree in Computer Science from the University of Colorado in 1989. Her research areas include distributed computations, modeling, and software engineering.

STEPHEN ELLIOTT is a software consultant. He has a B.A. degree in Music from Denver University in 1982 and an M.S. degree in Computer Science at the University of Colorado in 1984. His research interests include programming languages, networks, and software to support music composition.

JEFF McWHIRTER is a graduate student in the Department of Computer Science at the University of Colorado. He has a B.S. degree from Oakland University in 1986. His research interests are in distributed simulation and graph models of computation.

BRUCE SANDERS is a Professional Researcher in the Department of Computer Science at the University of Colorado. For the past two years, he has also held a teaching appointment in the Department, developing and teaching a senior-level software engineering projects course. He received his M.S. in Computer Science from the University of Colorado in 1978. Prior to joining the University, he was a Member of Technical Staff at Bell Laboratories and has held software engineering and management positions at NBI and Integrated Solutions. His interests include graphics, user interfaces, operating systems, modeling, and software engineering.

All authors' address is:

Department of Computer Science
Campus Box 430
University of Colorado
Boulder, CO 80309-0430
(303) 492-7581