Olympus Application Note: Capacity Planning with PN-Olympus

Gary J. Nutt†

CU-CS-522-91 March, 1991

Department of Computer Science
Campus Box 430
University of Colorado
Boulder, CO 80309-0430

† The author's telephone number is (303) 492-7581 and his electronic mail address is nutt@cs.colorado.edu.

ABSTRACT

Olympus Application Note:
Capacity Planning with PN-Olympus

Capacity planning tools are fundamental to sales and acquisitions organizations in vendor and user corporations, respectively. The sales organization is required to illustrate how the vendor's products can be used to meet the user's computational requirements. Alternatively, the acquisition organization relies on capacity planning tools to represent the need of the user corporation, and to purchase computational facilities that meet the need yet do not exceed the budget.

This technical report describes how PN-Olympus (an interactive, graphic modeling tool), can be used to support capacity planning for distributed computer systems. PN-Olympus is unique in its display of qualitative as well as quantitative behavior, and in its ability to support cooperative planning.

1. INTRODUCTION

For the past several years, researchers at the University of Colorado have been involved in the study of systems to support a spectrum of interactive, visual models of various aspects of distributed systems. The work has resulting in the development of a general architecture, *Olympus*, for such systems [14], and specific modeling systems intended to support specific application domains.

BPG-Olympus [13] is an instance of the architecture used to support a formal graph model with AND and OR control flow logic. Various small models have been constructed using BPG-Olympus, and Schauble has used the system and model to study CPU-memory interactions at the computer architecture level [18, 19].

Demeure used the Olympus architecture as a starting point for the *VISA* system for *ParaDiGM* models of distributed computations [3]. ParaDiGM is intended to represent process functionality and interprocess communication for medium-to-large grain distributed partitions, particularly of the type commonly employed in a network environment. VISA provides a modeling environment in which a designer can experiment with alternative strategies for partitioning the functionality of a computation into different processes, using different means for communication and synchronization.

Beguelin used the Olympus architecture, and the BPG-Olympus implementation as the basis of the *Phred* system for visual, parallel programming [2]. The Phred system takes advantage of the Olympus architecture to provide an asynchronous *critic* process to analyze a program for determinacy as it is being constructed.

Olympus systems are fundamentally modeling systems, supporting multiple user interactions with a single model. In this paper, we describe how another variant of the Olympus architecture -- one that supports interpreted Petri nets (called *PN-Olympus*) -- is used for multiuser capacity planning in distributed systems.

1.1. The Need for Capacity Planning Tools

Capacity planning tools are fundamental to sales and acquisitions organizations in vendor and user corporations, respectively. The sales organization is required to illustrate how the vendor's products can be used to meet the user's computational requirements. Alternatively, the acquisition organization relies on capacity planning tools to represent the need of the user corporation, and to purchase computational facilities that meet the need yet do not exceed the budget.

In the ideal world, the requirements of the sales organization and the acquisition organization are identical, i.e., to be able to analyze the user's computational requirements and to select a suitable configuration that meets that need.

Capacity planning has been an important aspect to sales and acquisition organizations for many years. The challenge is to derive a suitable representation of the load on a future system, then to predict the performance of various configurations under the speculative load. Such performance predictions allow the consumer organization to select a cost-effective configuration based on the predicted rate of growth in the load, budget, and

confidence in the prediction mechanism.

1.1.1. Analytic Models

Mathematical models are an essential tool for capacity planning. These models allow the capacity planner to represent the load using a mathematical description such as a mean interarrival time and a mean service requirement, or probability distribution functions describing the arrival and service patterns.

The basic queueing model consists of a queue and a server; it can be extended to a queueing network consisting of many queues and servers (and other resources). Such models can provided an indication of the expected performance based on a relatively modest amount of detail in the model and load descriptions.

There are several successful systems that allow a capacity planner to construct mathematical models of the load and system, then to solve the models to predict the performance, (e.g., see [7, 8, 17]).

The difficulty in relying exclusively on mathematical models and modeling tools is that one cannot generally achieve a reliable prediction of the detailed performance without adding corresponding detail into the model of the load and of the system. And as the amount of detail in the models increases, the mathematical model becomes more difficult to solve.

1.1.2. Simulation Models

Like other performance researchers, we advocate the use of simulation to complement mathematical modeling. A simulation model can incorporate an almost arbitrary amount of detail into the representation of the load and the system. However, a single run of the simulation on an individual set of parameters in the model may be misleading since it is merely one instance of predicted performance, not a representation of general behavior (as is the case with mathematical model results).

As a further step in capacity planning studies -- and in performance prediction in general -- we argue that a comprehensive system should provide a mechanism "between" mathematical modeling systems and traditional simulation systems. Our argument is built around the observation that as target systems (and the models) become more complex, the analyst's understanding of the behavior becomes jeopardized by the detail; the performance system should provide *qualitative* feedback relating to behavior in addition to the more traditional *quantitative* feedback of queueing and simulation systems. Once the analyst understands *how* the system behaves, then s/he can begin to experiment with the qualitative behavior.

As a guiding principle for computer supported modeling tools, the Olympus study adheres to the idea that *the system should always provide benefit that is at least proportional to the amount of effort to use the tool.* This means that if the model is difficult to construct, then it should provide large benefit; if it is easy to construct, then it may or may not provide large benefit. In capacity planning systems, this suggests that the

planner should be able to quickly ascertain performance predictions with low confidence, and that more effort should provide performance predictions with correspondingly higher confidence. We believe that the aspect of qualitative feel is an important aspect in the understanding of a system configuration, and is especially important in capacity planning.

1.2. Capacity Planning in Distributed Systems

Distributed systems aggravate the problem of capacity planning. Let us consider a simple family of distributed and parallel systems, namely ones that rely on a network to interconnect a set of von Neumann computers (i.e., we ignore vector machines, pipeline machines, shared memory multiprocessors, and exotic architectures in this discussion).

Even this characterization of distributed systems is more general than that with which contemporary commercial software (and hence capacity planning tools) rely. For example, contemporary distributed software typically does not use distributed virtual memory nor generalized cooperating sequential processes; instead, the software is built upon a model of sharing at either the disk level (e.g., diskless workstation systems), file level (e.g. file servers), or procedure interface (e.g., remote procedure call).

Consider the general form of contemporary distributed software: the software is partitioned into parts that communicate and synchronize using a message-passing paradigm. Because of the prevailing hardware environment to support parallel and distributed processing, the message-passing mechanism is implemented on a (local area) network. That is, Figure 1 is a characterization of the distributed computation.

The figure represents a general view of message-passing distributed software on a network of von Neumann computers. The distributed hardware platform requires that the programmer be knowledgeable about good strategies for breaking the software into pieces to be implemented on different computers, and to be familiar with programming tools to accomplish such distribution.

In the state-of-the-art, distribution strategies are not generally known -- and that was the thrust behind the VISA system mentioned earlier in the paper. That is, VISA is an interactive tool to be used by a programmer when s/he considers alternative strategies for

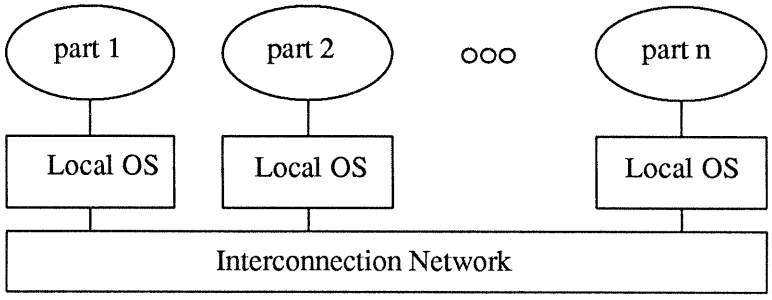


Figure 1: Implementing a Distributed Computation

partitioning the computation.

At the level of programming, most programmers are familiar with sequential programming techniques but are relatively unfamiliar with multi-threaded (multi-tasking or multiple process) computational models. This has encouraged system designers to hide the distribution under the abstract machine model presented to the programmer. Two general models are widely used in today's distributed software: the *client-server* model and the *remote procedure call* (RPC) model. Object-based systems may also be used for multi-threaded programming, but systems that implement distributed objects are still largely in the research world rather than in the commercial programming world. Thread-based programming models are also growing in popularity, but these too are not yet a proven commercial programming technology.

The client-server model of computation differs from the generalized model above as illustrated in Figure 2. In this model, each client is implemented by a single-thread process, i.e., a traditional sequential program in execution. Each process can share information (or other resources) maintained by the shared server process. The client process accesses the shared resource by making a procedure call on a procedure in the Local OS, which packages the request into a network message and sends it to the server. The server acts upon the requests, usually serially but possibly in an arbitrary manner. When the server has fulfilled the request, it typically returns a message to the client Local OS, which then performs an ordinary procedure call return to the client program.

Disk and file servers are obvious instances of the server model. The disk server client programs are actually parts of the operating system which correspond with the disk server

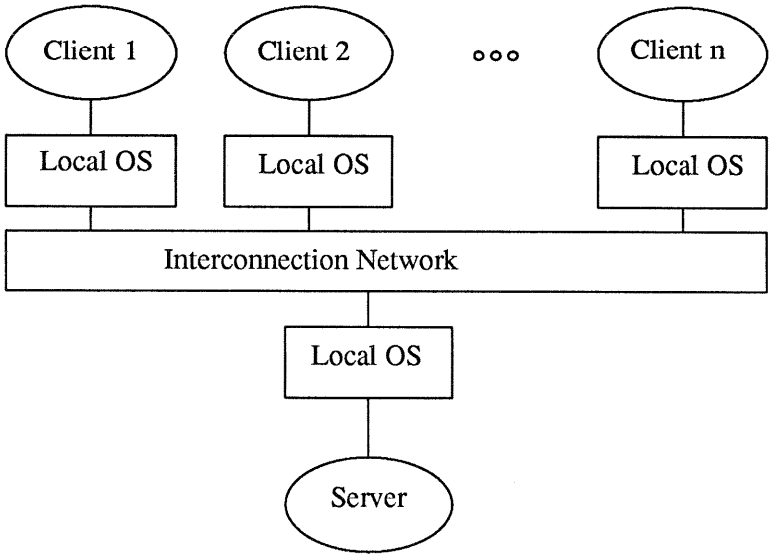


Figure 2: Client and Server Machine Configurations

via a common server interface rather than through a local disk driver. File servers differ from disk servers in that client programs make local calls related to file operations rather than disk operations. Similarly, communications servers and print servers provide a simple interface to the client programs -- one that looks much like an ordinary procedure call.

Remote procedure call is a refinement of the client-server model. Our explanation of the client-server model bears a strong resemblance to an ordinary procedure call model, i.e., the program is written to make a call on some Local OS system procedure, and does not continue execution until the server has responded to the request, allowing the Local OS to return from the system call. The client-server model allows more flexible forms of operation between the client and server, e.g., the client program could cause a request to be issued to the server without waiting for any response. In the RPC model, *only* the call-return model is allowed.

The client-server model typically requires that the services provided by servers be relatively static, shared across many clients. The RPC model supports the notion of user-defined remote procedures. That is, using RPC, a programmer can define a sequential procedure and have it be loaded and executed on a remote machine. Now, any authorized client can invoke the procedure with the procedure call synchronization scheme -- call and wait-for-return. The resulting sharing does not necessarily allow the caller and the callee to overlap their execution, i.e., there is no inherent parallelism, but there is distribution.

How can a capacity planning system address these new distributed computing paradigms? Both paradigms rely on the existence of a shared server and shared network communication. A bottleneck in the capacity of the system now may be in the client, the network, or the server. And capacity planning will grow from issues related to CPU speed and memory requirements to network and server sharing.

1.3. Capacity Planning as Computer Supported Cooperative Work

Capacity planning in complex systems benefits substantially if a team can be involved in the project, rather than having a single analyst performing the study. This suggests that capacity planning is an example of cooperative work; our approach uses the modeling system to implement an instance of *computer supported cooperative work* (CSCW) since we believe that the system can be most effective if it simultaneously supports several analysts working on a common planning model. Thus, we see the planning system as having the general form shown in Figure 3.

Each analyst is supported by a "Local Tool" facility to allow him/her to view and manipulate a shared planning model. Logically, these individual units coordinate the local view of the shared model that is maintained in a single global tool.

It is possible to incorporate various other shared tools in the global mechanism, but it is also easy to extend the shared functionality with particular tools implemented as "Extended Tools." An example of an Extended Tool might be an analyzer that checks a

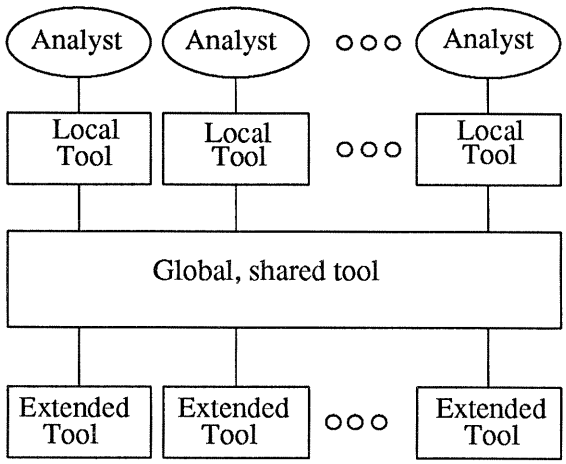


Figure 3: An Olympus CSCW System

Petri net (stored in the Global Tool) for liveness or safety. In a capacity planning system, an Extended Tool might be an MVA tool.

The advantage of the architecture illustrated in Figure 3 is that each block may operate asynchronously of the other blocks. For example, an Extended Tool can be analyzing a Petri net in parallel with editing and viewing in the various Local Tools. We have found this to be a very powerful aspect of the Olympus architecture, e.g., see the Phred critic [2].

This architecture suggests that there may be small delays between the time that a model is changed in the Local Tool and the time that it is displayed at all other local sites, but that there is a single global definition of the model.

The granularity of sharing among the local frontends is determined by the requirements of the modeling system. Ellis et al. describe the GROVE editor in which fine-grained activity, e.g., mouse movement, is reflected at all other local frontends [5]. A large-grained strategy might only update the local displays once every five minutes. The technical issues that arise in this context are related to locking protocols, cacheing, coherency and consistency, and IPC performance; we focus on those issues in our architecture documents rather than in application notes.

For effective shared capacity planning systems, we believe that the granularity should be at a relatively fine grain, e.g, that of user operation invocation, but not at the individual keystroke (or mouse movement level). Therefore, we define a set of atomic actions based on the semantics of the model, e.g., add/delete/move a component, then update local frontend knowledge at that granularity.

1.4. Using PN-Olympus for Capacity Planning

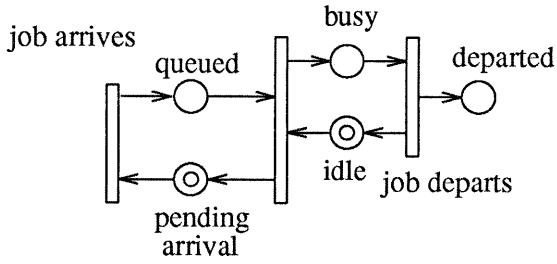
PN-Olympus is an instance of the Olympus architecture, designed to support interactive, visual modeling of interpreted Petri nets (cf. *information control nets* [4], *colored Petri nets* [9] and *predicate/transition nets* [6]).

In general, Petri nets have often been used to represent the activity of a computer system, e.g., see [10, 16]. In the capacity planning application, timed Petri nets [1, 11, 21] are more appropriate for observing performance (as opposed to synchronization in the general case).

As a simple example, consider the single-server queueing system shown in Figure 4(a). For this model to represent a system, the analyst must specify an arrival and a service pattern, e.g. as probability distribution functions, F_{arrive} and $F_{service}$. The Petri net model shown in Figure 4(b) represents the same system: the transition labeled "job arrives" represents the event of a job arriving; it can only occur if there is a token on the place labeled "pending arrival" (as illustrated in the figure). The job is assigned the server if and only if there is a token on the place labeled "idle" (as shown) and at least one token on the place labeled "queued". The Petri net represents performance by introducing "dwell time" for tokens on places. For example, one could use F_{arrive} to describe the dwell time for tokens on "pending arrival" and $F_{service}$ to describe the dwell time on the place labeled "busy". Performance characteristics of the Petri net model are reflected by the number of tokens on "queued", the distribution of time of tokens on "busy", and the number of tokens that were placed on "departed".

PN-Olympus is a distributed, interactive, visual modeling system (corresponding to the block diagram shown in Figure 3) used to construct, modify, exercise, and observe Petri

(a) Queueing Net



(b) Petri Net

Figure 4: A Queing Model

net models. It has been implemented on a set of Sun workstations, using the Sun SunView and NeWS windowing environments. PN-Olympus is itself a distributed program, i.e., the "local tool" parts of PN-Olympus run as client software on user machines while the "global and extended tool" parts of the system run on other machines as other clients and servers.

Each PN-Olympus user's local frontend system handles all human-computer interaction using the point-and-select model preferred by that user. The user may draw the model, then annotate it with various labeling and interpretation information. The resulting model is stored and interpreted by the global backend.

As a capacity planning tool, PN-Olympus allows the analyst to define the model and to animate it in scaled real time, providing qualitative feedback about the models performance with a low investment of effort. PN-Olympus also supports hierarchical definition, so that unwanted detail in the model can be abstracted into higher level constructs, e.g. a place in a Petri net can be modeled by another Petri net.

PN-Olympus allows the user to change the annotation, appearance, and state of the model at any time, including during animation or execution. We have found this to be especially useful during the phase of modeling in which the users (planners and customers) are building a qualitative feel for the behavior.

We refer the reader to the user's manual for further description of the use of PN-Olympus [12].

2. MODELING A DISKLESS WORKSTATION SYSTEM

A timely capacity planning application is the problem of configuring a diskless workstation environment. Diskless workstations are typically configured as shown in Figure 5, i.e., there is a set of *diskless client* machines and a shared *disk server* machine.† The software in a client machine is written as if it contained a local disk. The local operating system intercepts disk requests (at the driver level), reroutes them over the network to the server, accepts a response to the request from the server, then returns it to the local disk request.

The fundamental capacity planning question in a diskless workstation environment is usually something like "how many clients can a single server machine support?" And of course the answer to the question depends upon many factors, including:

- The nature of the load offered by each client
- The speed and processing capacity of each client
- The speed and processing capacity of the server
- The speed and processing capacity of the network

It is also natural to consider variations of the configuration, e.g., can two servers be used if the server is saturated?

---

† Architectural and implementation details of diskless environments can be found in the literature, e.g, see [15, 20].

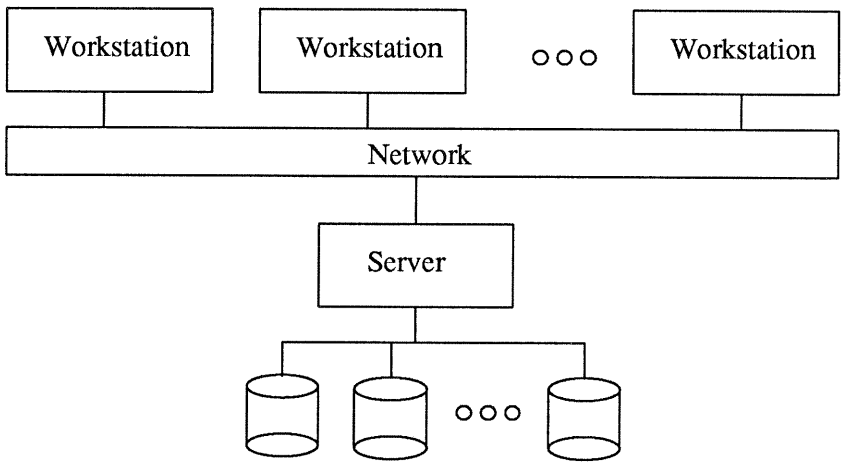


Figure 5: A Diskless Workstation Environment

In the remainder of this paper, we shall describe an example scenario for capacity planning using PN-Olympus.

2.1. A Simple Client-Server Model

The first task is to construct a representation of the load provided by a client machine, and to represent the server's activity. Figure 6 is a Petri net representing the disk requests by a client machine and activity of the server machine (cf. the queueing model above). The client part of the model simply generates tokens (representing disk requests), where the dwell time on the "thinking" place represents the interarrival time; it is clear that if we know a characterization of the arrival pattern as a probability distribution function, $F_{request}$, then we could use that distribution to define dwell time, and hence to emit tokens to the server part of the model. Similarly, $F_{service}$ can be used to model the time required for the server to honor a request by using it to define dwell times on "server busy".

The figure is a screen dump from the PN-Olympus system. In the state shown, a *dialog box* has been placed on the screen (by the user) to specify a normal probability distribution with a mean value of 3 and a variation of 1 (time units) for the place labeled "thinking", i.e., for the interarrival time of requests. These time units translate into seconds for animation.

Of course, the model only explicitly represents a single client and a single server. Our goal was to be able to explore the effect of adding clients to the system. There are two trivial changes we can make to the model to represent N clients rather than just one client: first, the mean time between disk requests could be divided by N, resulting in N times as many requests per unit time (on the average). Or the Petri net could initially be marked with N tokens on the "thinking" place in the model, essentially replicating that part of the model N times.

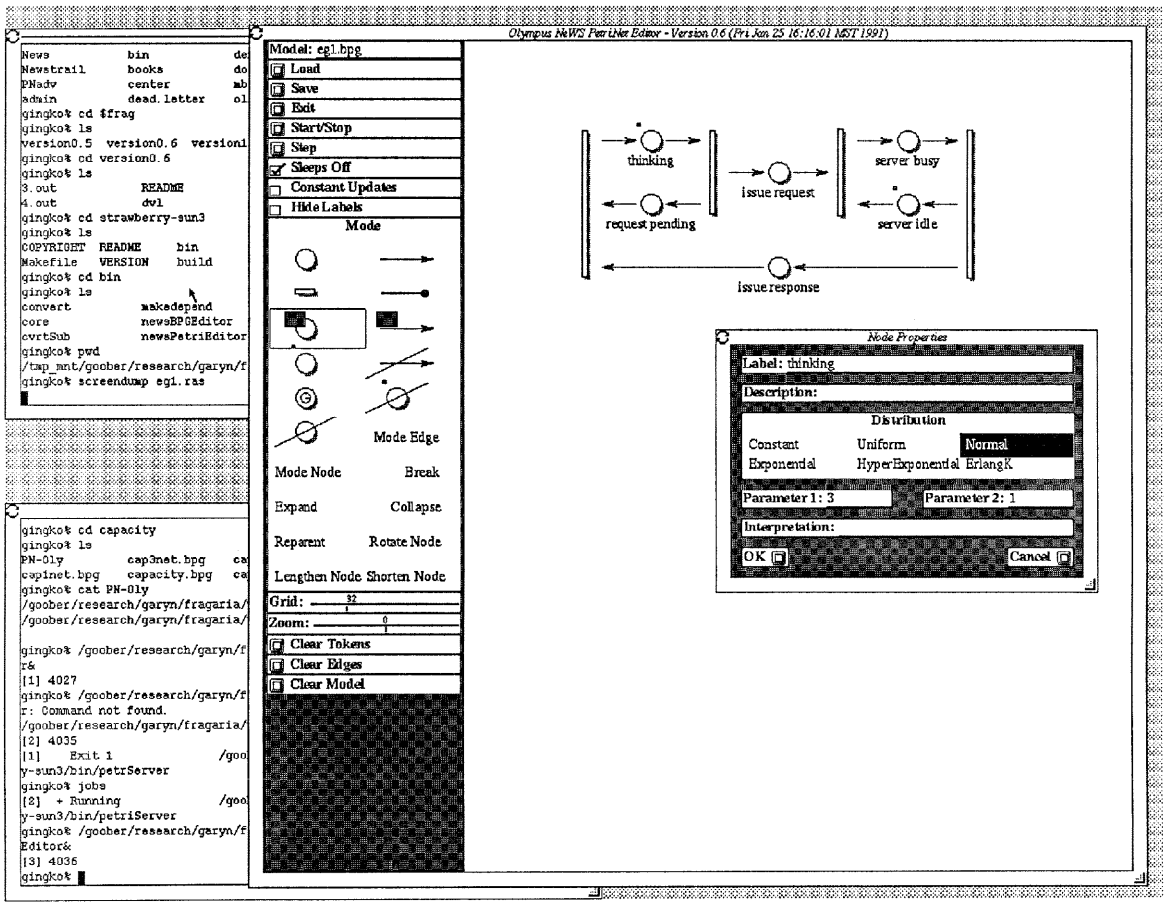


Figure 6: A Simple Client-Server Model

In PN-Olympus, both alternatives are trivial operations, the first requiring that the mean and variation parameters be edited in the property sheet for "thinking", and the second simply requiring that the planner use the mouse to place N tokens on the "thinking" place in the net. Further, the Petri net animation facilities make it easy for the planner to quickly perceive (and demonstrate) the qualitative behavior of the system -- based on the pattern of token movement -- under the specified load without ever considering distributions or tabular data. The qualitative behavior is based on animated activity in the model.

Following our guiding principle about the yield of result for effort expended, this model is economical to build, and it may provide us with considerable information about the effect of varying N.

But since the model is so simple, several factors are ignored, and they may be important within a particular installation. For example, the clients may not all behave in the same way, or the service time for a request may not be independent of the source of the request. To have a higher level of confidence in the observed behavior, the model can be

refined.

2.1.1. Refining the Client Model

First, consider a refinement of the model of the clients' behavior. A client machine may go through phases in which it uses the disk intensively, e.g., when it is formatting a document or compiling, and other phases in which it rarely uses the disk, e.g., when the user is editing text. The client model could be changed to reflect changes between the two phases as shown in Figure 7.† The mean value of the token dwell time for "I/O intensive thinking" is defined to be much smaller than that for "CPU intensive thinking".

Notice that if a token is returning from the server (as illustrated in the figure), it will enable transitions in both phases of the client subnet. This throws the use of the Petri net into a new realm: if the model is an "ordinary" Petri net, it is nondeterministic since even though the token on the "demultiplex" place enables transitions in both phases of the client, only one can actually fire, i.e., the token can go to one phase or the other.

PN-Olympus provides a facility to annotate each edge from "demultiplex" with a probability that the edge will be traversed, given that a conflict exists. Now, the model stochastically represents the amount of time that the client spends in each phase by establishing an appropriate probability that the token goes to the I/O or CPU intensive phase.

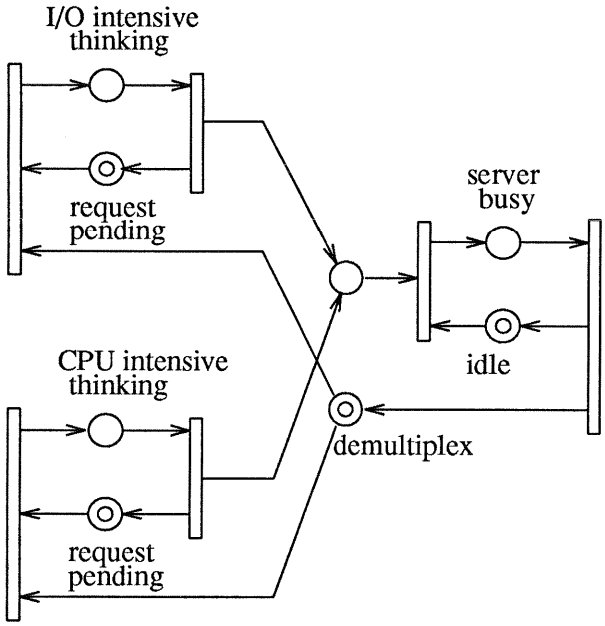


Figure 7: Representing Phases of Work

† The remaining figures in this technical report have been drawn with a drawing program, since the PN-Olympus graph editor is somewhat limited in its ability to label items, and to draw multi-jointed arcs.

However, this model does not really control the phases very well, i.e., each time a request is satisfied, the client may move to the next phase as determined by the probability. Thus we will tend to see random I/O and CPU activity rather than bursts of activity in either phase. PN-Olympus supports *interpreted* Petri nets in which a specific procedure can be called each time a token lands on a specific place. Suppose that the "demultiplex" place has the"demultiplex" procedure shown in Figure 8. Given that **select_out_arc** is able to resolve the conflict by routing the token along path 0 or 1 (corresponding to the I/O and CPU phases, respectively), then the network can be made to be deterministic (stochastic), and the client can be made to operate in one phase or the other for some number of iterations.

Now reconsider the case where we use multiple tokens on the client subnet to represent multiple clients. The net will not necessarily behave as we desire, since the different tokens from the different clients will tend to interfere with one another in the "demultiplex" part of the model. Of course this could be remedied by making the interpretation(s) be more sophisticated, but the planner could also simply refine the Petri net graph to explicitly represent more clients. We consider that approach below.

We note that this model continues to follow our guiding principle: the planner does little work and immediately gets a model that can be used for representing gross characteristics of the diskless environment.

2.1.2. Refining the Server Model

The server's behavior is essentially represented by the place labeled "server busy". We can use a probability distribution to represent variable time demand on the server, much as we used a distribution to represent the interarrival time of requests in the client.

```
static int iterations = 0;
static int phase = 0;       /* 0 means I/O phase, 1 means CPU phase */

demultiplex()
{
        if(iterations == 0)
        {
                iterations = sample(distribution(phase));
                phase = phase+1 mod 2;
        };
        iterations = iterations-1;
        select_out_arc(phase);
}
```

Figure 8: The 'Demultiplex' Procedure

However, suppose that the service of an I/O-intensive request should be modeled differently from a CPU-intensive request. There are two ways the analyst could proceed: the requests could be differentiated by an interpretation for "server busy" resulting in a procedure similar to the demultiplexing code. Or the "server busy" place could be refined into a subnet.

Figure 9 represents one possible interpretation to accommodate the need for different types of requests. This example illustrates one more important extension to Petri nets: when **server_busy** is called (i.e., when a token is placed on "server busy"), the token is assumed to have an *attribute* associated with it that identifies the "request_type" of the token (this attribute is assumed to have been set in procedures for "I/O intensive thinking" and "CPU intensive thinking". This attribute is used by **server_busy** to distinguish between requests from the I/O-intensive phase of a client from the CPU-intensive phase of the client.

Figure 10 is an alternative interpretation to the "server busy" place, namely as another Petri net with a single input and a single output. The "phase select" place will be required to demultiplex tokens when then enter the server, just as the "demultiplex" interpretation routed tokens to the appropriate phase in the client.

2.2. A Model of N Different Clients

With additional effort, the planner can expand the client server model to handle the N client case more intuitively and more accurately. Figure 11 is intended to represent the case that the two-phased client can be represented by an ellipse with "in" and "out" edges,

```
server_busy(token)
tokenDef *token;
{
        switch (token->request_type)
        {
        IN_OUT:         /* This token represents an I/O-intensive request */
                dwell_time = sample(distribution(I_O_service));
                break;
        CPU:    /* This token represents an I/O-intensive request */
                dwell_time = sample(distribution(CPU_service));
                break;
        default:
                break;
        };
}
```

Figure 9: The 'server busy' Procedure

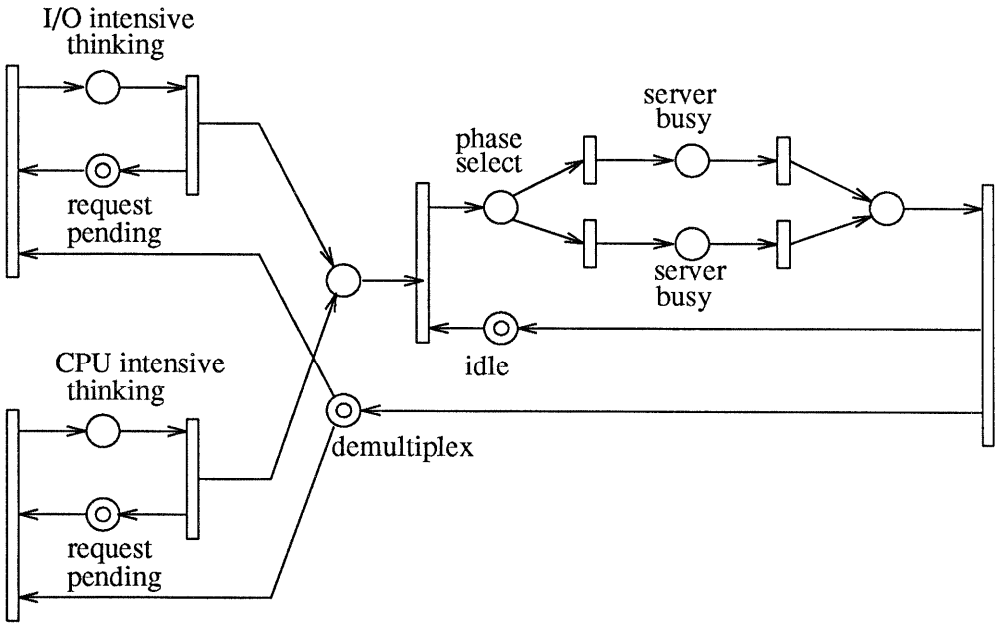


Figure 10: The Refined 'Server Busy' Place

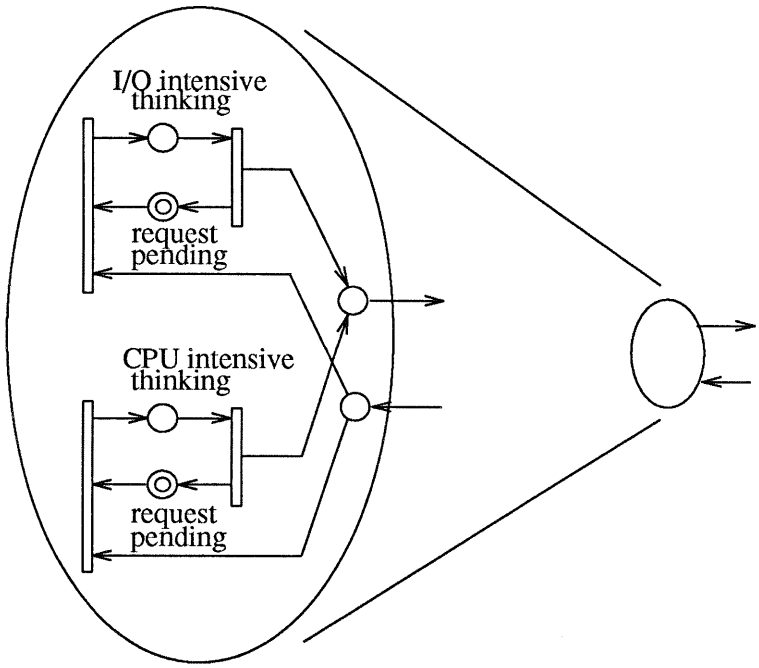
i.e. logically, every place that an ellipse appears in another Petri net, one could replace it with the subnet shown in the large ellipse in the figure. (This could be done by macro substitution, or by runtime links; PN-Olympus uses runtime links to support such *hierarchical* structuring of the model.)

The "issue response" place in this model serves a similar purpose to the "demultiplex" in the two-phased client model; it is intended to route a token back to the client that issued the corresponding request (rather than demultiplexing to different phases within a single client). We can again use token attributes to define a procedure for "demultiplex" as shown in Figure 12. The **map** function is assumed to map a **clientID** value into an arc identifier.†

The obvious benefit of the refined model is that it allows the planner to get a better qualitative understanding about the load on the server from the N clients, i.e., the simpler solutions shown above may not make some of this behavior as obvious. It also allows the clients to be defined differently, e.g. differing in the number and types of phases incorporated into each.

---

† The idea of a client ID leads directly to the notion of *colored* Petri nets [9]. This concept uses the notion that each token has a unique color; when a token passes through a transition that generates new tokens, then the children are the same color as the enabling token. And conversely, only tokens of the same color can enable a transition. In our case, we can nearly emulate a colored Petri net by writing a comprehensive set of procedural definitions for each "join" transition. Of course, if colored Petri nets are required for some set of applications, then it might be easier to incorporate color into the Olympus facilities with a Colored Petri Net version -- CPN-Olympus.



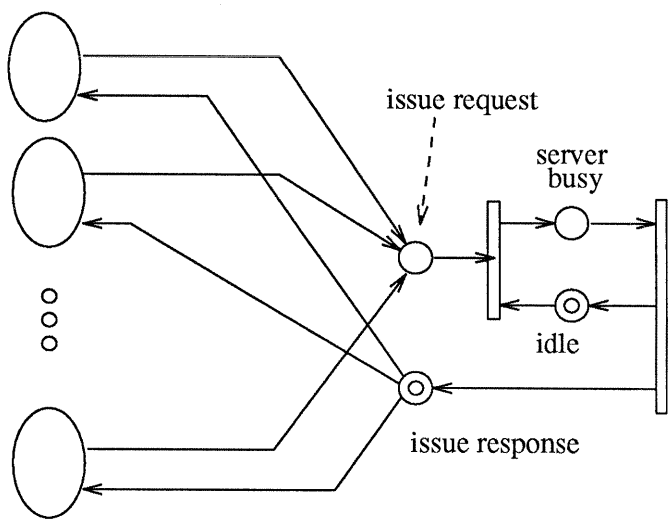


Figure 11: N Clients and 1 Server

```
demultiplex(token)
tokenDef *token;
{
        select_out_arc(map(token->clientID));
}
```

Figure 12: The Multi-Client 'Demultiplex' Procedure

PN-Olympus also includes an instrumentation facility. The planner can attach an instrument to any edge, transition, or place in the Petri net, then observe the number of tokens that pass across the model atom and the distribution of time that the token spends on the atom. For example, one could attach an instrument to the "server busy" place and monitor the server usage under the offered load from the N clients.‡

2.2.1. N Clients and M Servers

We can use a technique similar to the one for replicating clients to explicitly represent N servers; Figure 13 is a model with N clients and M servers. The place "issue request" ("issue response") is used to multiplex requests (responses) from (to) clients (servers) to servers (clients).

Demultiplexing a client request to a server may require more careful modeling than the use of the client identifier attribute of a token for "issue response". If all servers are identical -- any request can go to any server -- then the demultiplexing aspect of the associated procedure would reflect that policy. On the other hand, if the client has a favored server, then the procedure will select an output arc as a function of the client identification. Finally, if the servers provide specific functionality such that a request must be sent to a specific server, then the token will not only need to contain an attribute for the client identification, but also one to guide the token to the appropriate server. The former policies would probably be used for a set of name servers, but the latter policy would be used for our disk server example.

2.3. Introducing the Network

As the number of clients and servers increase, the network will become an increasing factor in the overall performance (particularly in those cases where there is substantial protocol overhead in using the network). Figure 14 is another refinement to incorporate the network into the model.

The model of the network is simple (just as the model of a client and a server are each simple); the "net idle" place represents the case in which any client or any server can use the network to transmit information. Any client uses the network whenever it places a

‡ Each instrument preserves observed data in a file; the obvious method of presenting this instrumented data is in tabular form. We have prototyped visualization mechanisms for viewing the data collected by the instrument as the model runs. Essentially, we have found that a small window that plots the utilization vs time -- updated as the model executes -- (similar to Sun Perfmeters) to be quite effective.

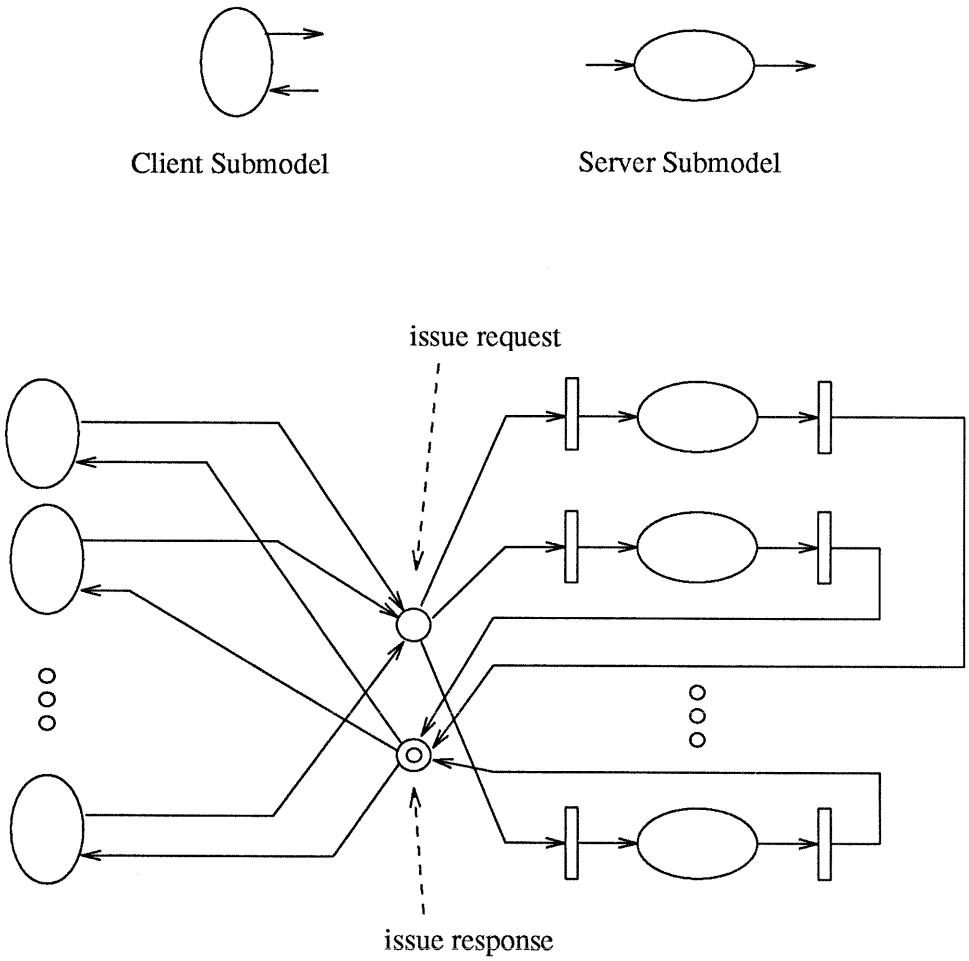


Figure 13: N Clients and M Server

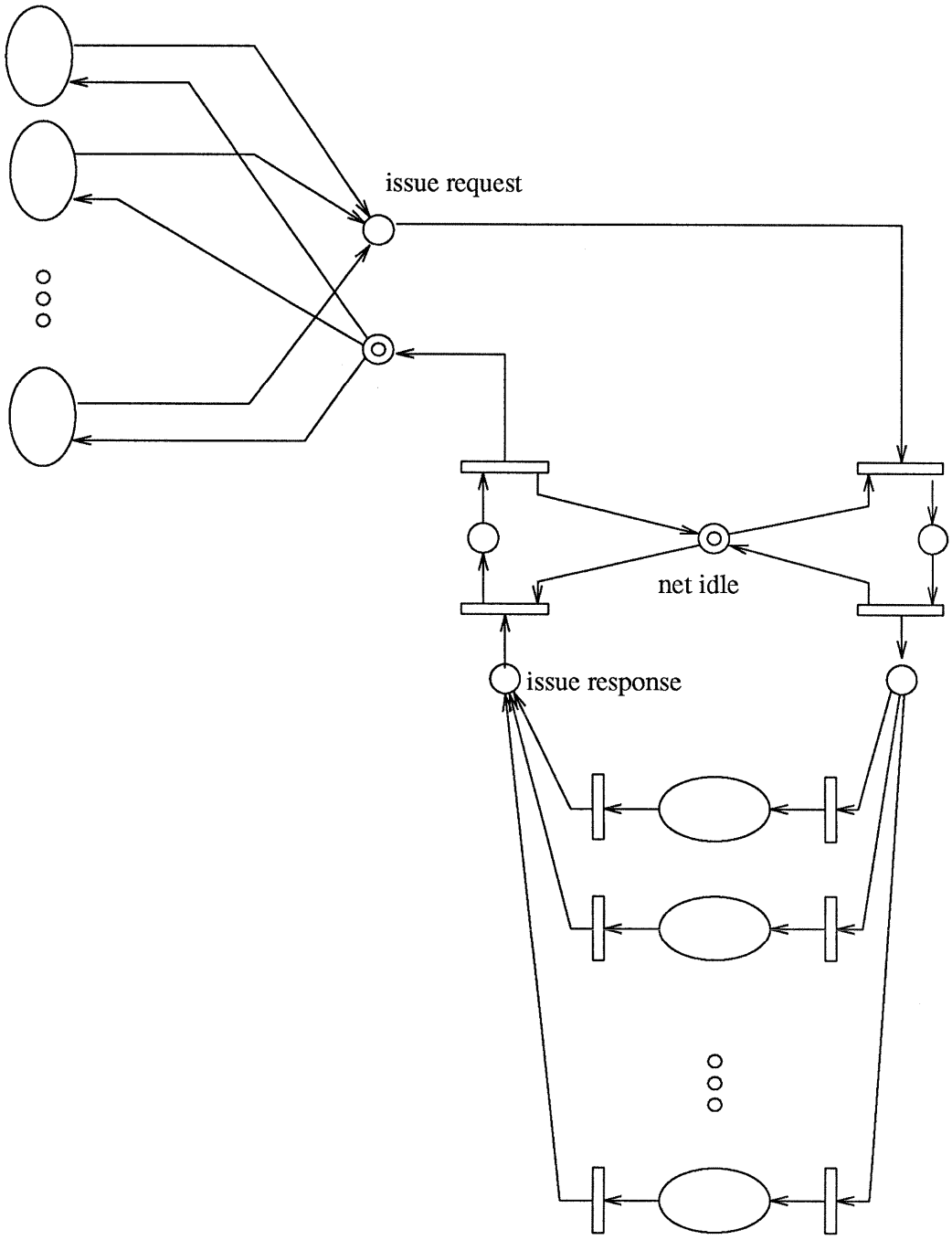


Figure 14: Incorporating the Network

token on "issue request" and any server uses the network when it places a token on "issue response". The subnet represents the time which the shared medium is allocated to a sender; protocol overhead would be modeled with an appropriate dwell time (or other simulation definition) at the two issuing places.

3. CONCLUSION

Contemporary network configurations have become far too complex to determine without capacity planning tools. We have attempted to illustrate how Olympus-based systems can be used to support capacity planning by providing an extended scenario of how PN-Olympus might be used to study a diskless workstation environment.

We advocate the use of a spectrum of tools, and illustrate how PN-Olympus can be used quickly obtain a qualitative understanding of the network behavior. PN-Olympus models can then be refined to act as full simulation models as the need for more detailed analysis becomes relevant. PN-Olympus has been designed to provide comprehensive tools to support the guiding principle that the tool should provide results that are at least proporational to thc amount of effort expended by the user.

Our approach embraces a spectrum of performance prediction technologies, ranging from analytic modeling tools, through animation, to simulation. We believe that it is necessary to provide a full spectrum of tools so that simple models can be constructed and used quickly, while refined models can be built from the simple models, providing more detailed performance information.

Finally, although we have not emphasized the multi-user aspect of PN-Olympus in the application note, we believe that as the models grow more sophisticated it becomes increasingly important for the modeling tool to support cooperative observation and modification of the model. This cooperation might take place between a planner and a customer -- located at different sites, or among a group of planners and customers. If performance tools remain in the domain of single user tools, then the value of cooperative problem solving and of qualitative observation of the performance cannot be utilized to their full extent.

This note has described an *application* of our research program. Our research into interactive modeling systems continues, driven by the state-of-the-art in systems and software research, and by the results of application of the technology to real world problems. We strongly encourage the use of most of our prototype systems; interested parties may contact the author to obtain a copy of the PN-Olympus or the BPG-Olympus systems.

4. BIBLIOGRAPHY

1. G. Balbo and G. Chiola, "Stochastic Petri Net Simulation", *1989 Winter Simulation Conference Proceedings*, Washington, D. C., December 1989, 266-276.
2. A. L. Beguelin and G. J. Nutt, "Collected Papers on Phred", University of Colorado, Department of Computer Science Technical Report CU-CS-511-91, January 1991.
3. I. M. Demeure and G. J. Nutt, "Collected Papers on VISA and ParaDiGM", University of Colorado, Department of Computer Science Technical Report CU-CS-488-90, August 1990.
4. C. A. Ellis and G. J. Nutt, "Office Information Systems and Computer Science", *ACM Computing Surveys 12*, 1 (March 1980), 27-60.
5. C. A. Ellis, S. J. Gibbs and G. L. Rein, "Groupware: Some Issues and Experiences", *Communications of the ACM 34*, 1 (January 1991), 38-58.
6. H. J. Genrich, "Predicate/Transition Nets", in *Petri Nets: Control Models and Their Properties, Advances in Petri Nets 1986, Part 1*, W. Brauer, W. Reisig and G. Rozenberg (editor), Lecture Notes in Computer Science, Springer Verlag, Berlin, Heidelberg, New York, 1987.
7. *PAWS/GPSM marketing brochures*, Information Research Associates, Austin, TX, 1988.
8. J. T. Jennings and C. U. Smith, "GQUE: A Tool Using Workstation Power to Aid Software Design Assessment", draft technical report from Performance Engineering Services, Austin, TX, 1988.
9. K. Jensen, "Coloured Petri Nets", in *Petri Nets: Control Models and Their Properties, Advances in Petri Nets 1986, Part 1*, W. Brauer, W. Reisig and G. Rozenberg (editor), Lecture Notes in Computer Science, Springer Verlag, Berlin, Heidelberg, New York, 1986, 248-299.
10. T. Murata, "Petri Nets: Properties, Analysis and Applications", *Proceedings of the IEEE 77*, 4 (April 1989), 541-580.
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, M. Beaudoin and N. Sloane, *Petri-Olympus User's Manual*, Department of Computer Science - University of Colorado, Boulder, August 1989 (revised December, 1990).
13. G. J. Nutt, A. Beguelin, I. Demeure, S. Elliott, J. McWhirter and B. Sanders, "Olympus: An Interactive Simulation System", *1989 Winter Simulation Conference Proceedings*, Washington, D. C., December 1989, 601-611.
14. G. J. Nutt, "A Simulation System Architecture for Graph Models", in *Advances in Petri Nets*, Springer Verlag, to appear, 1990.

15. G. J. Nutt, *Centralized and Distributed Operating Systems*, Prentice Hall (in preparation), 1990.

16. J. L. Peterson, *Petri Net Theory and the Modeling of Systems*, Prentice-Hall, Inc., 1981.

17. C. H. Sauer, E. A. MacNair and J. F. Kurose, "The Research Queueing Package: Past, Present, and Future", *AFIPS Proceedings of the National Computer Conference*, 1982.

18. C. J. C. Schauble and L. D. Fosdick, "Simulation of Parallel Computations", *Proceedings of the Symposium on Scientific Software*, June 1989, 65-66.

19. C. J. C. Schauble, "A Memory Access Simulator for MIMD Machines", University of Colorado, Department of Computer Science, PhD proposal, April 1989.

20. D. Swinehart, G. McDaniel and D. Boggs, "WFS: A Simple Shared File System for a Distributed Environment", *Proceedings of the Seventh Symposium on Operating Systems Principles*, December 1979, 9-17.

21. M. K. Vernon, J. Zahorjan and E. D. Lazowska, "A Comparison of Performance Petri Nets and Queueing Network Models", Technical Report #669, Computer Sciences Department - University of Wisconsin, Madison, September 1986.