ON IMPLEMENTING THE OSI MODEL IN ADA VIA TASKING

Norman R. Howes
Department of Computer Science
George Mason University
4400 University Drive
Fairfax, Virginia

Alfred C. Weaver
Department of Computer Science
University of Virginia
Charlottesville, Virginia

The purpose of this paper is to investigate the performance limitations imposed by the overhead associated with the use of the Ada language when implementing the OSI Reference Model as proposed in Buhr [1]. The investigation was conducted on both sequential and parallel Ada implementations. This study led to the introduction of a new proposal that introduces concurrency in a fundamentally different way than that introduced in the Buhr proposal. This new architecture incurs significantly less overhead than the Buhr proposal.

ADA, COMMUNICATIONS, AND CONCURRENCY

There have been recent proposals for implementing the International Standards Organization's Open Systems Interconnect (OSI) Model in Ada using the Ada language feature for concurrency called a task, as for instance [1] and [2]. Also, there have been a number of recent articles detailing the performance of Ada tasking that show the performance of this feature of the language is highly implementation dependent, as for instance [3] and [2].

The purpose of this paper is to investigate the performance limitations imposed by the overhead associated with the use of the Ada language when implementing the OSI Model as proposed in [1]. Buhr's proposed architecture can take advantage of a multiprocessor Ada implementation but, as will be shown, the associated overhead is significant. Also, a new proposal will be advanced for implementing the OSI Model via tasking that introduces concurrency in a fundamentally different way than in Buhr's proposal. This new implementation can also take full advantage of a multiprocessor Ada implementation while incurring significantly less overhead than the Buhr proposal.

The investigation documented in this paper took place in two phases. The first phase took place at the NASA—Johnson Space Center where the authors were engaged in the development of a prototype version of the network operating system for the Space Station program. During this phase the authors had no access to a parallel architecture Ada implementation. Consequently, the first phase consisted of investigating the Ada overhead associated with the Buhr proposal in a sequential processing environment. Comparisons were made between the overhead on a Digital Equipment Corporation VAX 11/785 and a Rational 1000. This led to the introduction of a new architecture that is considerably more efficient than the Buhr model and it was conjectured that the new model would continue to be more efficient in a parallel environment. This phase concluded with an investigation of the performance implications of abandoning concurrency; i.e., of using only procedures to implement the OSI Model in Ada for single processor machines and the construction of a linear mathematical model that could predict the Ada overhead for various architectures on single processor machines.

During the second phase, the authors investigated the problem in a parallel environment using a Sequent 821 with eight processors. It was possible to configure the Sequent environment to use a variable number of processors ranging between one and seven. This enabled the benchmarks to be run in a variety of parallel environments. This study verified the above conjecture for the Sequent implementation. Furthermore, the proposed new architecture is of the dispatcher/server type so the ability to reconfigure the Sequent environment permitted the study of the relationship between the number of processors and the number of servers. The investigation of the proposed new architecture thus becomes a two dimensional problem that is somewhat more complex than the single processor investigation. For that reason the results are organized in two sections, the first being the sequential case and the second the parallel case.

THE BUHR MODELS

In his book "System Design with Ada", R. J. A. Buhr [1], proposes several "models" for designing layered communications software using Ada tasking to decouple the layers. All of these models are similar in that they introduce concurrency in essentially the same way. Each layer is implemented as an active package (one that contains tasks) and the tasks in a given layer rendezvous with tasks in the layer packages above and below it. The first model that Buhr introduces on page 187 is the simplest and also introduces the least overhead. For this reason, it will be the only one dealt with in this paper since is can be considered the "best case" for Buhr's approach.

The presentation of what will be referred to as "Buhr's first model" will be in terms of an icon-oriented diagraming technique introduced by Buhr that is both elegant and efficient, and which is a fundamental part of his design methodology. In the discussions that follow, these "Buhrgrams" will be used to illustrate the concepts. Buhr's first model is illustrated in Figure 1.

In this diagram, the large rectangles denote Ada packages whereas the small rectangles inside the large ones represent procedures. Those procedure rectangles that touch the outer package rectangles are the procedures that are visible from outside the package (i.e., included in the package specification). The parallelograms inside the package rectangles represent tasks. The big arrows indicate calls to either procedures or tasks. The direction of these arrows indicate which program unit is calling which other program unit. The head of the arrow indicates the program unit that is being called whereas the tail indicates the calling unit. The smaller arrows with little circles at the tail indicate the parameters that are being passed. The dots at the head of some of the calling arrows indicate the presence of guards; i.e., the task entry is conditional.

In Buhr's first model, all calls between layers originate from the higher layer. The advantages of this model according to Buhr are (1) it is intuitively simple, (2) requires no more rendezvous than his other models and (3) is structurally free from constructs that could lead to deadlock in one form or another.

Each OSI layer in this model is implemented as a package. The task labeled M in the middle of each package rectangle is the main task for that layer whereas the tasks labeled T are transport tasks as defined by chapter three of Buhr or in Nielsen [6]. From Figure 1 it can be seen that it requires a minimum of two rendezvous to "pass" a message from a higher layer to a lower one or vise-versa. It also requires at least one

procedure call. The procedures in this model are labeled P in Figure 1.

The main task does whatever processing is required at the layer in which it resides and thus could conceivably have subtasks or procedures. The transporter tasks are merely the decoupling mechanism that allows the main task to work asynchronously from tasks in adjacent layers without having to periodically poll them or be polled by them to see if a message is ready to be passed between layers. In practice, an implementation would probably only pass the protocol header associated with the layer rather than the entire message.

Buhr's First Model

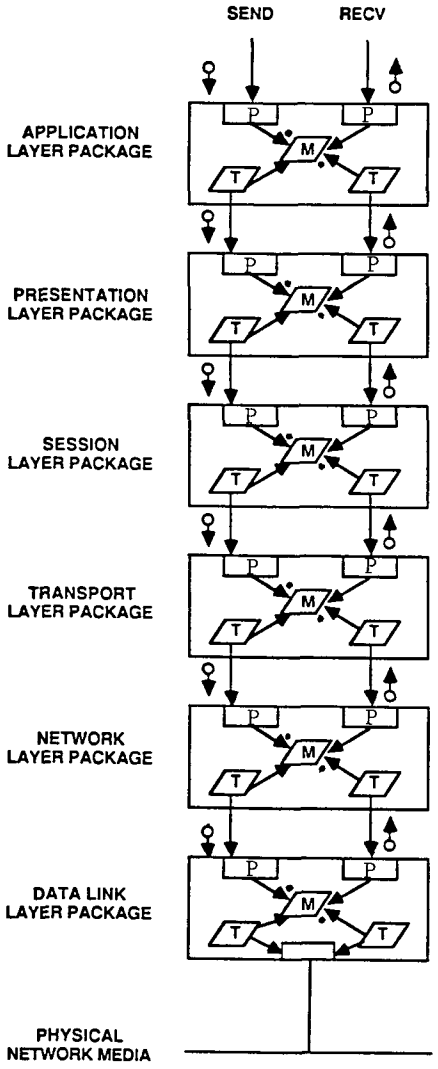

Figure 1.

THE SEQUENTIAL CASE

The benchmarks to determine the Ada overhead for implementing the OSI seven layer stack were run on a Digital Equipment Corporation (DEC) VAX 11/785 and a Rational 1000. It was determined that a procedure call took approximately 19.2 μsec on the VAX 11/785 and 4 μsec on the Rational 1000. The overhead associated with a simple producer-consumer rendezvous where the only parameter passed is an integer proved to be 835 μsec on the VAX 11/785 and 51 μsec on the Rational 1000. As pointed out in [2], for tasking paradigms that employ multiple tasks in various roles (e.g., transporters, buffers, relays, etc.) the Ada overhead associated with the entire paradigm is not necessarily the sum of the number of rendezvous multiplied by the overhead associated with a simple producer-consumer rendezvous.

Consequently, to measure the Ada overhead associated with the Buhr models, it was necessary to construct benchmarks that simulated the entire model. To simulate Buhr's first model, it was decided to pass integers from one layer to another using the architecture depicted in Figure 1. The main task (M) in each layer package merely passed the integer on to the main task in the next layer via a transporter task (T). In effect, this made each layer a null layer since the main task did nothing but pass the message (integer) on. By passing an integer, it prevented the compiler from optimizing the rendezvous away and also provided a convenient mechanism for determining which message was being handled by which layer package at any given point in time.

Extensive tracing was done by identifying consecutive messages with consecutive integers and dumping a map of which message was being serviced by which layer package at any given time. This was done to try to gain an understanding of the impact of the task scheduling algorithms employed on the different single processor machines on which the benchmarks were being conducted. The authors were in possession of a DEC Ada compiler evaluation that indicated that rendezvous times would increase substantially as the number of concurrent rendezvous were simulated on the underlying single processor machine and expected that this tracing would shed some light on the phenomenon. However, the authors were unable to duplicate these reported observations. It was observed that the DEC Ada default task scheduling algorithm (run until blocked) ran a little faster than when using pragma TIME_SLICE. It is these default algorithm times that are reported in this paper.

It should be clear that this method of simulating Buhr's first model only simulates the Ada overhead associated with it. The times reported in this paper should therefore be understood to be lower bounds on the actual messaging rates that could be expected if this model were to be implemented on one of the machines on which the benchmarks were run. The Ada overhead associated with an implementation of Buhr's first model on a VAX 11/785 proved to be 11.2 msec per message whereas the overhead associated with a Rational 1000 implementation was 650 μsec per message.

Expected Message Delay Times for Buhr's First Model

In what follows the expected length of time E(t) between the arrival of a message at the seven layer OSI stack and its departure will be investigated. Specifically, that component of the expected delay time attributable to the Ada overhead will be calculated for both the VAX 11/785 and the Rational 1000. For these calculations, the customary assumption of Poisson arrivals is made. The equation for E(t) is given by:

$$E(t) = 1 / \lambda E(n) \qquad (1)$$

where λ is the *mean message arrival rate* and E(n) is the expected length of the incoming message queue. A simple derivation of (1) can be found in [ ]. The formula for E(n) is:

$$E(n) = \frac{\rho}{1-\rho} \qquad (2)$$

where ρ is the *traffic intensity* given by the ratio λ/μ where μ is the *mean service rate*. The mean service rate is defined as the inverse of the time it takes to service a single message. For the VAX 11/785 the time to service a single message attributable to the Ada overhead was found to be 11.2 msec. For the Rational 1000 it was 650 μsec. Consequently, for the VAX 11/785, if the Ada overhead was the only thing contributing to a message's delay time then μ would be approximately 89 messages per second and for the Rational 1000 it would be approximately 1,518 messages per second. With these values for μ it is possible to plot the following graphs of lower bounds of E(t) with respect to mean message arrival rate for both the VAX 11/785 and the Rational 1000.

A New Model

As depicted in Figure 2, the message handling rate for Buhr's first model is going to be fairly slow on the VAX 11/785 just based on the overhead associated with concurrency management in Ada. On the other hand, Figure 2 indicates the feasibility of implementing a message handling system based on Buhr's first model on the Rational 1000 with quite respectable performance. This, of course, is due to the fact that the rendezvous times on the Rational 1000 are an order of magnitude faster than on the VAX 11/785.

Lower Bounds on Expected Message Delay Times Attributable to Ada Overhead for Buhr's First Model

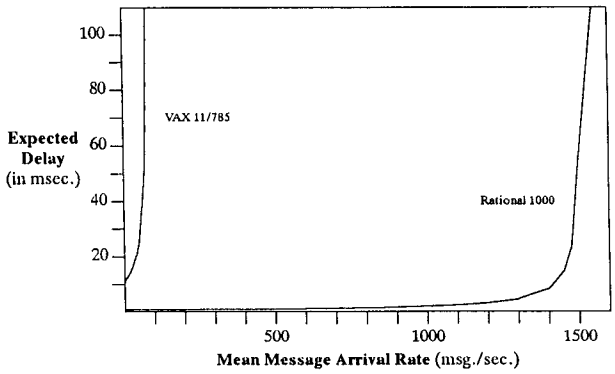

Figure 2.

The Rational 1000 is a very specialized machine whose hardware architecture was designed expressly for exploiting the Ada language features. It was designed to be a system to host an Ada Programming Support Environment (APSE) for developing Ada code for a variety of other target machines. At the time, the cost of a Rational 1000 was considerably greater than the cost of a VAX machine that could accommodate a similar number of general purpose users. For these reasons, it is unlikely that the Rational 1000 would be used as the basis for a message handling system. The benchmarks were conducted on this machine merely to show that the overhead associated with implementing a message handling system based on Buhr's first model is highly dependent on how the Ada language system is implemented rather than any intrinsic deficiency of the language itself.

At the time of the writing of this paper, there are over 120 Ada compilers that have been validated. Probably more of these compilers have been validated on VAX machines than on any other. Consequently, they represent a class of machines that are likely candidates for implementing message handling systems in Ada given the DOD's mandate for the use of Ada and the large number of VAX networks currently in service. The question naturally arises: Is it possible to implement a respectable OSI Model-based message handling system in Ada on a VAX?

There are two approaches to be investigated here and both will be dealt with in this and the following sections. First, concurrency could be abandoned; i.e., the message handling service could be designed in such a way that procedures are used in place of tasks. This is a natural approach since the processing of message headers in the OSI Model is intrinsically a sequential process. There is no reason to require a simulated parallel processing of the six layer headers (on a single processor machine) as in Buhr's models when the header for layer six cannot be constructed until the header for layer seven has been completed and so on down the layer stack.

But there are some disadvantages to abandoning concurrency. First, by abandoning concurrency, there will be no way of exploiting concurrency in the event the message handling system could be run in a multi-processing environment at a later date without redesigning the system. This is worthwhile considering, given the current trend in computer architectures. Secondly, it is conceptually easier to express certain useful design features such as the ability of higher priority messages being able to bypass lower priority messages in the message processing stream by having concurrent streams for different priorities.

The trick is to design a message handling system that exploits concurrency but does not suffer such a high overhead as the Buhr models. Such a model will now be proposed. Using the above observation that the processing of message headers in the OSI Model is an inherently sequential process, it can be seen that Buhr's models are not very suitable for modeling this process. They are too rendezvous laden because they introduce concurrency at too low a level. It takes at least 14 rendezvous to get a message "through" the OSI stack.

Consider instead the architecture depicted in Figure 3. Here, a package is depicted for handling the process of sending messages in an OSI Model message handling system. This package provides the capability for building the OSI Model message headers and transmitting them. The package consists of a dispatcher task, a family of server tasks and a transmitter task. The dispatcher task receives all outgoing messages. Figure 3 depicts a situation where all the server tasks do exactly the same thing; i.e., the dispatcher task merely provides a buffering function and all the server tasks call the same entry to get the next available message for processing. This diagram could be changed a little to depict the situation where the server tasks represent different message processing streams for different priority level messages. In this case, the different server tasks would call different entries in the dispatcher task. One can easily imagine other scenarios where the server tasks could provide other distinct capabilities.

Each of the server tasks would then deposit their messages (complete with the various OSI Model layer headers) in a buffering transmitter task that would be asynchronously emptying the transmit buffer onto the network. Each server task would process an entire message as opposed to the philosophy of the Buhr models where different tasks are required to process the different layer headers.

It can be argued that the Buhr model is more modular and therefore more in the spirit of the OSI Model. Since each layer of the OSI stack is implemented as a package, any one of the layers could be replace with one that employed a different protocol simply by replacing the corresponding package. But the modularity of the Buhr design can readily by transferred to the new model. For this, one need only implement a header processing package that contains an inactive package (no tasks) for each layer. This package can then be imported into each server task.

The New Model

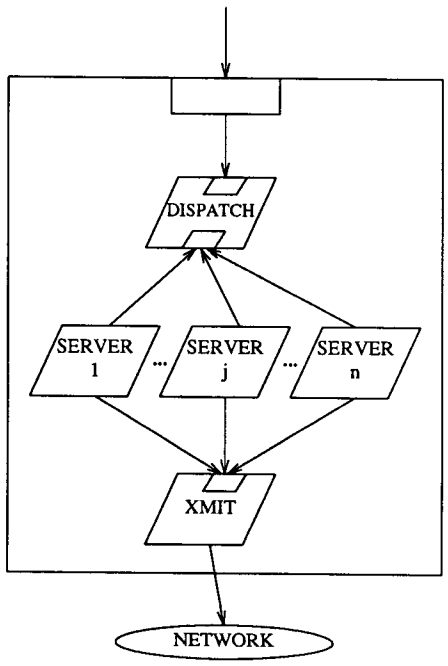

Figure 3.

The Ada overhead associated with the implementation of the new model on the VAX 11/785 proved to be 2.6 msec per message while the overhead associated with the Rational 1000 implementation was 139 μsec per message. Again, if we assume that the Ada overhead is the only thing contributing to a message's delay time then for the VAX 11/785, the mean service rate μ will be 384 messages per second while for the Rational 1000 the mean service rate will be 7,194 messages per second. With these values of μ we can plot the following graphs of lower bounds of the expected message delay times attributable to Ada overhead with respect to mean message arrival rate for both the VAX 11/785 and the Rational 1000. These graphs are shown in Figure 4.

From these plots, it can be seen that this new model has much more respectable message delay times attributable to Ada overhead than the Buhr models. At least theoretically, the possibility exists of developing (via the new model) a 200 message per second (2,000 byte message) system with delay times of under 50 msec including transmission delays. A message handling system of this performance is about the minimum that could be considered "real-time" for a broad class of military and industrial applications.

It should also be noticed that if a machine like the VAX 8800 were used, or a better optimized Ada compiler on the VAX 11/785, the expected delay times should be reduced to the very respectable range for the implementation (via the new model) of a real-time message handling system based on the OSI Model. Of course, the total delay times in any implementation are going to be very dependent on the functionality of the various layers no matter what the implementation language.

Also, it should be noticed that with the Rational 1000 machine, the Ada overhead associated with the new model is almost negligible. This shows the potential of the language for real-time message handling systems when the Ada rendezvous is highly optimized. There exists at least one flight qualified computer with an Ada implementation that claims rendezvous times under 100 μsec. If this is the case, similar performance to that observed on the Rational 1000 could be expected. Such an architecture with very small computers and fast rendezvous should make feasible very high performance distributed systems based on an OSI Model message handling system as are being investigated for the Space Station [4].

Lower Bounds on Expected Message Delay Times Attributable to Ada Overhead for the New Model

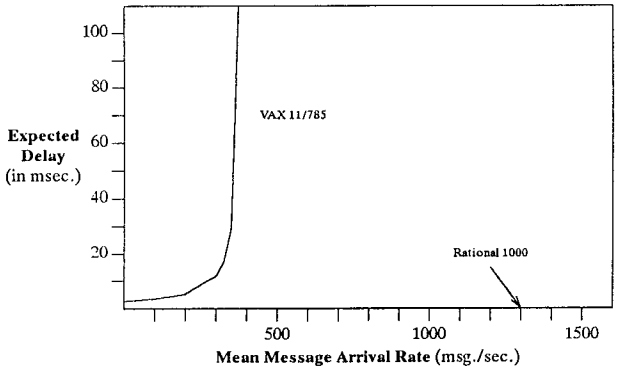

Figure 4.

ABANDONING CONCURRENCY

It is also interesting to consider to what degree the Ada overhead can be reduced by not using the Ada language feature of tasking. If it is desired to optimize the throughput of a message handling system to be implemented on a single processor machine then one must consider processing the messages sequentially as fast as possible. If the package for processing OSI message headers consisted of one procedure for each layer, the architecture of such as system might look something like that shown in Figure 5.

The Ada overhead associated with implementing a totally sequential (no tasks) message handling model as shown in Figure 5 on the VAX 11/785 is 165 μsec per message, while on the Rational 1000 it measured 51 μsec per message. In other words, the non-tasking model on the VAX would have about the same Ada overhead as the new model implemented on the Rational 1000 which is essentially negligible. Although this approach drastically reduces overhead on the VAX 11/785, it is clear from the previous section that it should not be necessary for most applications.

Ada Package with One Procedure per Layer

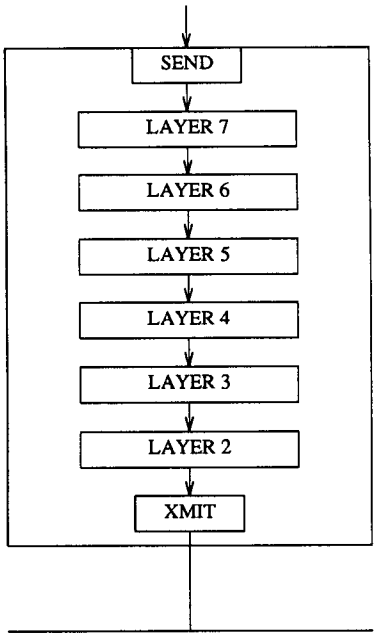

Figure 5.

MATHEMATICAL MODELS

An elementary mathematical model can be stated for all of the OSI Model implementations that have been discussed in this paper. It is often useful to have a simple mathematical model that can be used to evaluate a number of preliminary models before constructing benchmarks for each one of them. This enables one to narrow the field to the more promising ones. Notice that from the benchmarking results of the previous sections that the time τ to process a single message is approximately:

$$\tau = M \cdot R + N \cdot P \tag{3}$$

where M is the number of rendezvous needed to pass a message from layer seven to the physical network, R is the overhead associated with a simple producer-consumer rendezvous, N is the number of procedure calls needed to pass the message from layer seven to the network and P is the overhead associated with a procedure call. Since $\mu = 1/\tau$ we have from (1) and (3) that:

$$E(t) = \frac{M \cdot R + N \cdot P}{1 - \lambda(M \cdot R + N \cdot P)} \tag{4}$$

for a single processor machine. This mathematical model of an OSI Model implementation gives close approximations for plotting E(t) with respect to λ for all of those investigated in this paper. Equation (4) should also be valid for Ada implementations where the rendezvous times for all the rendezvous needed to get a message from layer seven onto the network are similar to the time required for a simple producer-consumer rendezvous.

THE PARALLEL CASE

The Buhr model behaves in a rather curious fashion on the Sequent, or perhaps it would be more nearly correct to state that the Sequent handles the Buhr model in a curious way. At first glance the results for the Buhr model appear to be predictable. The Ada overhead for a single message is approximately 29.5 msec for the single processor configuration. This reduces to about 19 msec for the seven processor configuration. Although considerably higher than the overhead on the VAX 11/785 and the Rational 1000, the overhead goes down as the number of processors increases. A graph of the overhead per message plotted against the number of processors is shown in Figure 6.

Overhead Associated with the Buhr Model for a Single Message on the Sequent

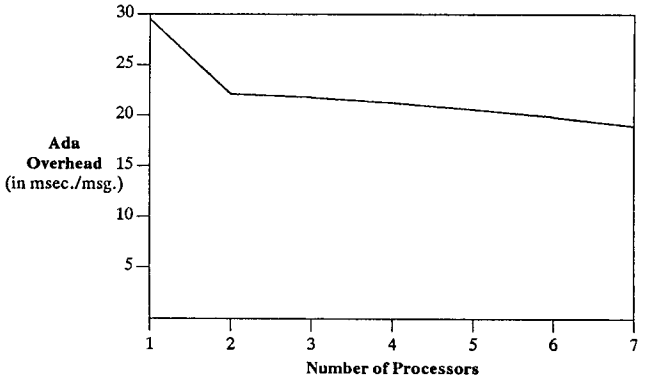

Figure 6.

But, a little reflection should convince one that as the number of processors increases, the overhead should go up. This is because as the number of processors increases, concurrency management becomes more complex. Total processing time may decrease if the application is compute bound, but the overhead should increase. That this is the case is confirmed by a number of benchmarks that will be discussed below.

The new model behaves quite differently than Buhr's model on the Sequent. There is a variable present in the new model that is not present in the Buhr model, and the Sequent Ada implementation is somewhat sensitive to this variable. The variable is the number of servers in the new model. In the Buhr model, the Ada overhead varies with respect to the number of processors, whereas in the new model it varies with respect to both the number of processors and the number of servers. Since the computation of the overhead for the new model on the Sequent is a two dimensional problem, a large number of benchmarks had to be run to determine the overhead associated with the new model and a family of curves were generated, one for each server configuration. This family of curves is shown in Figure 7.

Overhead Associated with the New Model for a Single Message on the Sequent

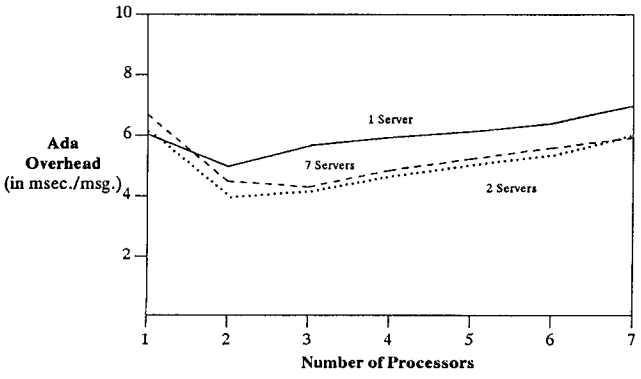

Figure 7.

As predicted, the Ada overhead for the new model proved to be significantly less than the Ada overhead for the Buhr model on the Sequent. In fact, the difference is even more pronounced on the Sequent than on the two sequential architecture machines tested. As can be seen from Figure 7, the best results were obtained with precisely two processors and two servers. Every other combination exhibited greater overhead. Another rule that can be observed from Figure 7 is that the best result for a given processor configuration is always achieved when the number of servers is less than or equal to the number of processors.

Finally, the procedures-only model yielded a result that was only fifty percent better than the best result obtained with the new model on the Sequent. In summary, the best result obtained for the Buhr model on the Sequent was about 19 msec, the best result for the new model was approximately 4 msec, and the best result for the procedures-only version was 2 msec. The corresponding lower bounds on the expected message delay times attributable to Ada overhead for these models are shown in Figure 8.

Lower Bounds on Expected Message Delay Times Attributable to Ada Overhead for All Three Models

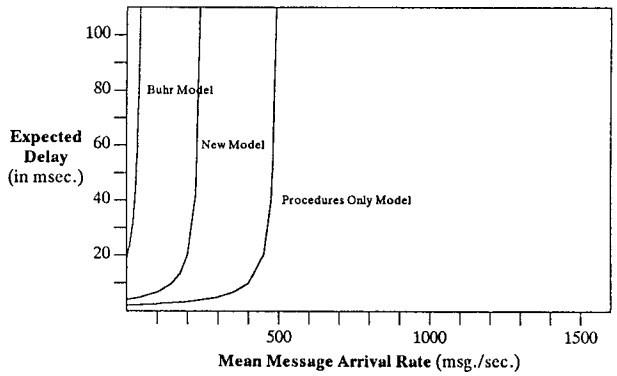

Figure 8.

It may be of interest to note the overhead associated with an Ada procedure call and with a simple producer/consumer rendezvous on the Sequent. The overhead associated with a procedure call was 51 μsec and did not vary with respect to the number of processors utilized. The overhead associated with a simple producer/consumer rendezvous increased with the number of processors. It was 1.55 msec for a single processor configuration and increased to 2.22 msec for the seven processor configuration. This result makes the timings for the Buhr model on the sequent even more curious. With so many rendezvous and with the rendezvous times increasing with the number of processors, one would think that the overhead associated with the Buhr model would increase rather than decrease as the number of processors increased.

CONCLUSIONS

From the foregoing discussions, it can be seen that the performance of a communication system based upon the OSI model is dependent upon a number of factors and that the range of performance is very broad over each of these factors. Unless careful consideration is paid to these factors and the sensitivity of the performance to these factors on a given implementation is quantified as part of the preliminary design of the system, a performance envelope could be entered that would render the performance goals of the system impossible.

In spite of the extensive differences in performance on these three fundamentally different machine architectures, there are some similarities that are worth noting. The most obvious is that on all of the architectures, the new model always performed considerably better than the Buhr Model and that the procedures-only model performed at least twice a well as the new model. This fact has significant implication for the the design of communication systems and probably for their first cousins, real-time systems.

The important design lesson to learn here has to do with the purpose of concurrency. Concurrency in a design is often advocated for the wrong reason. There are various situations in which concurrency can be an advantage, but in each of these situations, how concurrency is introduced makes a big difference. Concurrency that may enhance one situation may well degrade another, even though the other situation may benefit from a different kind of concurrency. This is probably best illustrated by three examples. For a compute bound algorithm on a parallel architecture machine, concurrency may well be a goal in itself. Design goals like minimal overhead (efficiency) and simplicity of the algorithm may be sacrificed to achieve minimal total execution time.

Next consider a situation in which concurrency is introduced to conceptually simplify a design that would be more complex otherwise. For instance, a priority messaging system might be expressed in a conceptually simple manner by using parallel tasks to model the handling of messages of different priority classes. As the third example, consider a concurrent design that is based strictly on real world concurrency; i.e., concurrency is only introduced to handle physically concurrent processes. An example of this might be a message handling system where messages are coming from multiple physically distinct terminals.

It is sometimes mistakenly thought that one and the same method of decomposing a system into concurrent tasks can be used for all of the above goals. While this may be true from a functional point of view, it is manifestly untrue when performance is taken into consideration. Any concurrency paradigm for the first two examples will probably produce a design that is far too rendezvous laden if applied to the situation of the third example. In the second example, concurrency is introduced for conceptual clarity. In Ada, this will usually lead to "unnecessary" tasks such as *guardian tasks* to guarantee exclusive access to critical resources or *transporter tasks* to allow asynchronous processing. These "unnecessary" tasks do not correspond to physical concurrency in the real world.

Similarly, in the first example tasks are introduced wherever concurrency can bring more processors to bear on the computation simultaneously which is a step even further away from physical concurrency. In the design of communications systems where performance is usually a primary consideration, concurrency needs to be introduced into the design only where it is used to model physical concurrency in the problem domain.

It can be seen that the New Model adheres to this principle. Individual messages exist concurrently in the real world. At any instant in a message's life, its form may differ from its form at another instant. For example, the message may begin as a simple string of characters. Thereafter, the layer seven header is added, then the layer six header, and so on. After the message has been transmitted, it may exist at another node, but all through this process it still maintains a physical identity; i.e., it is still the same message. The New Model assigns an Ada task to each of these physically concurrent objects.

The Buhr Model violates this principle. It assigns tasks to the processing of pieces of physically concurrent objects, namely, the headers associated with the messages. As a result numerous "unnecessary" transporter tasks are introduced into the architecture that do not correspond to physical concurrency. This results in significant additional overhead without contributing to the ability to bring multiple processors to bear on physically concurrent processes.

REFERENCES

1. Buhr, R. J. A., *System Design with Ada*, Prentice-Hall, Englewood Cliffs, New Jersey, 1984.

2. Burger, T. M. and Nielsen, K. W., *An assessment of the Overhead Associated with Tasking Facilities and Task Paradigms in Ada*, Ada Letters, Volume VII, No. 1, 1987.

3. Burns, A., Lister, A. M. and Wellings, A. J., *A Review of Ada Tasking*, University of Bradford, Computer Science Report, PR. 12, 1986.

4. Howes, N. R. and Raines, G. K., *A Simulation of the Space Station Computer Network*, ACM & IEEE Symposium on the Simulation of Computer Networks, Colorado Spring, August 1987.

5. Maki, D. P. and Thompson. M., *Mathematical Models and Applications*, Prentice-Hall, Englewood Cliffs, New Jersey, 1973.

6. Nielsen, K. W., *Task Coupling and Cohesion in Ada*, Ada Letters, Volume VI, No. 4, 1986.

7. Schultz, W. L., Bae, I. D., Chandna, A. and Khatibi, F., *Implementing a Factory MAP Network Simulation Tool Using the Ada Programming Language*, Workshop on Factory Communications, March 1987.