A Toolset for Performance Engineering and Software Design of Client-Server Systems

Greg Franks* Alex Hubbard* Shikharesh Majumdar* John Neilson†
Dorina Petriu* Jerome Rolia* Murray Woodside*

February 18, 1998

Abstract

TimeBench/SRVN is a prototype toolset for computer-aided design and performance analysis of software, with an emphasis on distributed client-server systems. The performance behaviour of such systems may defy intuition because it involves factors in the software design (such as the partitioning of the functionality and the frequency with which requests will be made to each server) and in the configuration of the distributed system (including replication of services, the distribution of data, and the speed of network access). The novelty of the tool consists in providing support both for developing design specifications and also for performance analysis. The integrated approach avoids the semantic gap between a designer's domain and the performance modeling domain, and assists the designer to explore factors that impact the performance of a design.

The performance models are based on the Stochastic Rendezvous Network (SRVN) formalism for client-server systems with synchronous service requests. The distinctive features of SRVNs are nested services (since servers can also act as clients to other servers) and the existence of two or more phases of service (the first executed while the client is blocked, and the others executed in parallel with the client).

TimeBench/SRVN is intended as a demonstration of the concept of an integrated designer/performance interface, and as a research environment for fast analytic solvers for the models. Besides a simu-

*Department of Systems and Computer Engineering, Carleton University
†School of Computer Science, Carleton University

lation solver, it offers three approximate analytic solvers based on recent research, a Markovian solver, a technique for finding bounds on the throughput without too many assumptions, and a tool for rapidly exploring the space of possible parameter values.

1 Introduction

The client-server paradigm for distributed computing is becoming increasingly popular. It poses its own characteristic problems for design, many of which are performance related. A software design can have performance consequences which are not intuitively obvious, due partly to having many interacting components and resources, and partly to the way that delays compound each other during a response. The service time of a software server may depend on a lower level server and may be increased dramatically by congestion at the lower level. Early decisions such as partitioning of functions between clients and servers determines these effects, and at an early stage before the system is deployed. Their impact can only be evaluated by modeling.

The process and the interface for building and interpreting models should be easy to use. Any description that captures the software structure and behaviour can also be the basis for a performance model. However, the existing modeling tools tend to be dominated by the performance aspects and by concepts which are not natural to software developers. This is particularly obvious in the Petri-net approaches described by Ibe, Choi and Trivedi [12], by Buchholz [3] and by Franceschinis and Muntz [11]. Tools such as RESQ [26] or (to a lesser extent) SES Workbench [28], which employ queueing models, really require a front end to translate a software model into queueing system parameters. Such a front end is provided by Smith [30], who has a fully-developed method for performance engineering of many kinds of software. It begins with a flow-chart-like description which gives a familiar interface, but which must be defined outside the normal effort of development. Thus, it is extra work which must be justified every time it is done. Our goal is to capture most of the information from a routinely-used development tool. Vetland and Hughes have also considered this point for software similar to that considered here, with a methodology and a design tool, but have not integrated a performance analyzer with it [31].

This research was part of a broader attempt to create a true computer-aided design environment for concurrent software with capture, analysis (including performance), interpretation and transformations to good-quality code. It produced a tool called TimeBench [6] based on the design methodology expounded in [5]. The performance component of TimeBench is described here. It accepts a software description, prompts for performance parameters and a specification of the environment for operation, and constructs and solves models to give performance predictions. It presently includes five different model solvers.

The present paper concentrates on software for distributed systems with a multi-layered architecture in which tasks interact by one making a request for service from another. This puts them in a client-server relationship, so we call them "client-server systems" (noting that they are more general than two-layer systems with just "client tasks" and "server tasks"). This simple but powerful paradigm is at the basis of network operating systems, remote procedure call (RPC) [1] systems, transaction processing systems, and the standards currently being elaborated in distributed computing, such as DCE (Distributed Computing Environment) [9] from the Open Software Foundation. It is natural to show these systems as layered networks, as in Figure 1. There are typically several layers of tasks, with lower layers offering more generic system services and upper layers offering more application-specific services, as illustrated in the Figure.

In these client-server distributed systems the performance problems arise from overloading of devices (processors, storage devices, network links, routers, etc.), overloading of software server tasks (where requests may queue for service), overloading of other resources such as locks, synchronization delays waiting for a parallel request to complete, and long compute times (including overheads). A single request to a remote server incurs operating system overheads, network delays and server contention delays. It is important not only to study these delays but also to estimate correctly the number of requests that are made. For example, partitioning of services can lead to more interactions than expected.

Layered queueing models which follow the layered software structure are used here. They model hardware and software resource contention uniformly. A "service" offered by a server task is in general a sum of components due to service at various lower level servers, including the hardware. The important abstractions are the counts of individual lower-layer requests and the time for each (with a breakdown for components due to the network, the server and its queue, etc.). In some cases there is also a classification of service types offered by a single task into operations by different

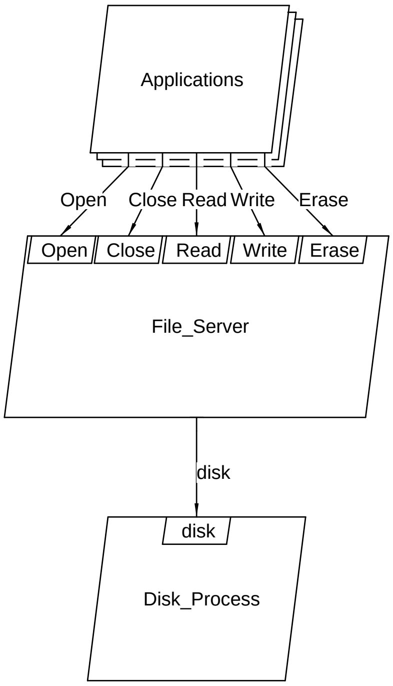

Figure 1: A three layer client-server model. The large parallelograms represent tasks and hardware devices while the smaller enclosed parallelograms are service entries.

"entries", as discussed below. Several authors have used an alternative approach for these systems, with timed Petri Nets, as mentioned above. Broadly speaking, these other models identify the same sources of performance loss and utilize the same input parameters. However, their solvers consider system states in greater detail and tend not to scale up well. Solvers which use versions of approximate Mean Value Analysis tend to scale up better, as described in Section 4.

The remainder of the paper describes the modeling abstractions and an example in Section 2, and the TimeBench tool interface in Section 3. Section 4 discusses the evaluation of performance by a variety of solvers which are integrated into the tool. Section 5 describes an example showing the modeling capabilities for a simple system, and Section 6 has conclusions.

2 Software Design and Performance Models

This section describes two differing views of a computer system, one from the perspective of a software designer, and the other from the perspective of a performance modeler.

2.1 Software Designers' View

Software designers normally regard their systems as collections of interacting software components such as tasks, modules and procedures running on a set of one or more processors. These components interact with the physical world in order to accomplish some function in a timely manner. To formalize the design process, various authors have created notations describing the entities and their relationships. Two examples are by Booch [2] and Buhr [4]; the latter is called MachineCharts and is the basis of TimeBench. A MachineChart is a directed graph in which nodes represent components such as tasks, modules and procedures and arcs represent inter-task communication and procedure calls. Table 1 defines the most-used graphical objects.

Figure 1 uses these symbols for a client-server system with three layers: some applications, a file server, and a disk process. The arcs all represent synchronous request-wait-reply interactions. (MachineCharts can also include an asynchronous one-way interaction.) For this discussion we assume that the designer is at the stage of determining what the distinct concurrent components should be and how they should interact. Functional modules are identified and associated with

Nomenclature	Icon and Meaning
Task	A parallelogram representing a single autonomous sequential thread of control (a task or process)
Task Set or Multi-server	A set of identical copies of a task, sharing a common processor and message queue
Arc, or Request Arc	A directed arc representing a request-wait-reply interaction (e.g., an Ada Rendezvous)
Service Entry	A small parallelogram on the boundary of a task representing an interface or port giving access to a particular service offered by the task (e.g., an Ada entry)
Procedure	A rectangle with an elongated side representing a reentrant procedure or function
Module	A rectangle representing a reentrant module (e.g., an Ada package)

Table 1: Nomenclature

particular tasks; a related cluster of functions may be placed in a single task. The interface and services offered by the task to its clients are identified and a service entry is included for each distinct service. Points at which requests are to be made are identified.

Hardware devices can also be shown directly in the MachineCharts model. For example, in Figure 1, a disk device could be added as a fourth level server. However, adding devices in this fashion would quickly clutter the diagrams, so task-device associations are usually made through textual annotations.

Many of the components in Buhr's notation can be refined by specifying internal behaviour through additional levels of diagram. For example, Figure 2 shows the internals of the file server task in Figure 1. The service entries from Figure 1 appear again at the top, to show how they relate to the internal structure. There is also a textual annotation language which is not used for performance analysis, and hence not described here.

2.2 Performance Considerations in the Design of Client-Server Systems

The high-level design decisions discussed above can have a profound effect on performance. The boundaries of partitioning the functionality between tasks determine the balance of task loading and also the degree to which the load can later be balanced across processors. Similarly, the scalability of

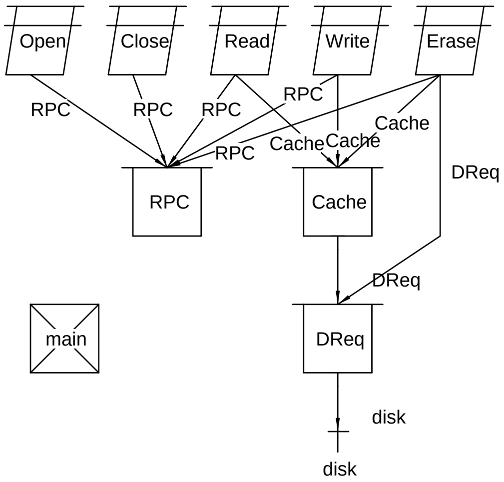

Figure 2: Refinement of the interior of the task labeled `File_Server` in Figure 1. The parallelograms in this figure represent service entries. The push-button-like icons are procedures or functions, and the arcs are procedure calls or messages.

the design is affected; this refers to the ability to scale up to more users by distributing, replicating some key tasks. To evaluate these effects at this early stage requires the ability to look ahead to execution within various physical environments and allocations, which can be obtained by modeling. Also, a model can determine the vulnerability of performance to potential changes, for instance in the number of users, the execution time of some module, or the number of service requests needed to complete a function. This gives early warning of sensitive areas.

One of the features of these systems which makes modeling valuable is the "software bottleneck" phenomenon investigated in [19]. When the service time of a task includes time for requests to other tasks, the task may be saturated when other resources and devices are not. We have found this to be impossible to identify without a model, since delay at an upper level depends on queue lengths at lower levels. One way to relieve a bottleneck is to run multiple copies of the sensitive task, another is to change its request pattern.

The model which follows was developed specifically to address this class of system and their problems.

2.3 The Stochastic Rendezvous Network Performance Model

The Stochastic Rendezvous Network (SRVN) [32, 33] or Layered Queueing Network [24, 25] performance model describes systems that use *synchronous service requests*, a request-wait-reply style of inter-task communication where the sender is blocked until the receiver replies. The SRVN model is special in that it incorporates the notions of *phases* and *included service*, both of which are described in greater detail below. Neither phenomenon is handled by well-known queueing network solution methods such as product-form Mean Value Analysis (MVA), although the method of surrogate delays [13] has been applied to included service. Fontenot [10] has also described a method for dealing with a single server with included service.

The core of an SRVN model is a directed graph whose nodes are service entries and whose arcs represent visits (requests) from one entry to another. A task is a group of entries, and is assigned to a processor. Although SRVN tasks are implicitly single-threaded, a multi-threaded task with identical threads can be modeled as a multi-server and a task with heterogeneous threads can be modeled as a group of separate tasks running on the same processor. Tasks which receive no requests are pure clients and are referred to as *reference tasks* (for example, the tasks labeled *Applications* in Figure 1); the primary model result is the throughput at each of these entities. Tasks which make no requests themselves to lower levels are pure servers. Tasks at intermediate levels act as both clients and servers.

A task is assumed to cycle infinitely, accepting requests and executing them. Each request causes the execution of an entry. In each cycle the task executes its own entry procedure and makes requests to other tasks. Reference tasks have a single implicit entry and loop forever without requiring input; they generate the system load.

The execution of an entry is divided into *phases*. Phase one is that part between receiving a request and sending the reply; it is absent in a reference task (because there are no requests) or in cases where the reply is just an automatic acknowledgment of the request (as in a pipeline task). Phase two is the part after the reply, executed concurrently with the requester. A third phase is useful to describe the passing of data to the next stage in a pipeline of tasks, so the tool accommodates three phases.

Included service refers to the time a task is blocked waiting for a reply after sending a request to a lower level server. It includes the message transfer delay, queueing delay and the service time itself.

The parameters of a task include execution times (the total average execution time of each phase of each entry), and visit ratios (the average number of requests from each phase of each entry to other entries). Software performance parameters must be either drawn from experience (perhaps using the systematic approach suggested by Vetland and Hughes) or estimated by the analysts using a process similar to that described by Smith [29]. Additional performance parameters describe the intensity of the load (i.e., the number of users), and the configuration of the system (such as the number of servers, the distribution of requests across disks and the number and speed of processors).

A comprehensive set of average performance measures are computed by the solvers (described in Section 4), including task and processor throughputs and utilizations, and mean queue lengths and delays. Values by entry can also be obtained if desired, and may help to pinpoint a performance problem. We have not made use of distributions and percentile values, but they can be provided by the simulation solver.

3 The TimeBench/SRVN Toolset

The goals for this toolset were, first that it should be an augmentation to a tool that can be used for software design, so that all the software structure and behaviour can be captured from the design description; and second, that it should be easy to add the performance-specific information and to obtain and interpret the results. We limited our goals to client-server systems, and to doing the analysis at a fairly high level, essentially at the level of communicating tasks.

These goals led to the following principles which guided the project. First, the modeling paradigm was made to align very closely with the domain of application. In a client-server system, the parts and parameters of an SRVN model can be traced directly back to the components of the software. This makes it easier for a designer to define the parameters and to understand the significance of the results. When a result shows a long queue at a task, this shows that the components giving the service are heavily loaded, for example. Second, the performance parameters and results should be presented in the context of the design, so it is immediately obvious which component gives rise to a problem. Third, the procedure to enter the model parameters is as automatic as possible and is coupled to the graphics.

3.1 Design Specification Aspects

TimeBench is a graphical tool (a more complete description is found in [6]). Components are created, arranged and connected by operations with a mouse. Figure 3 shows the interface in the midst of an operation to add an arc while constructing the example of Figure 1. The controls have features which are convenient and powerful, although they are not our main concern here. Powerful features include structured hierarchical definition of subcomponents within modules, the visibility of behavioural specifications within a structural context, execution of the specification with animation and tracing, and targetable code generation.

3.2 Performance Parameter Specification

To place parameters in the context of the design, a parameter form is associated with each graphical object. The form for the set of tasks labeled *Applications* is shown in Figure 4. Also, the most important parameters for each object are displayed on the screen attached to the object's name (in Figure 4 we see the value 0.0 for the execution time on *Applications* and 1 for the visit ratio on each arc). Performance results are similarly displayed in the form and on the screen, to give their context.

To help make data entry faster and more complete, the tool will sequence through all the objects in a system and present the form for each one in turn. Data are also checked for completeness before solving and an error list is presented (visible near the top of Figure 4). If an item in the error list is selected, the corresponding screen is displayed and the object is highlighted or shaded (as in the *Applications* task in the Figure). Also, default values are supplied for some parameters. For example, a separate processor is defined initially for each task; the processor allocation and scheduling can then be edited later.

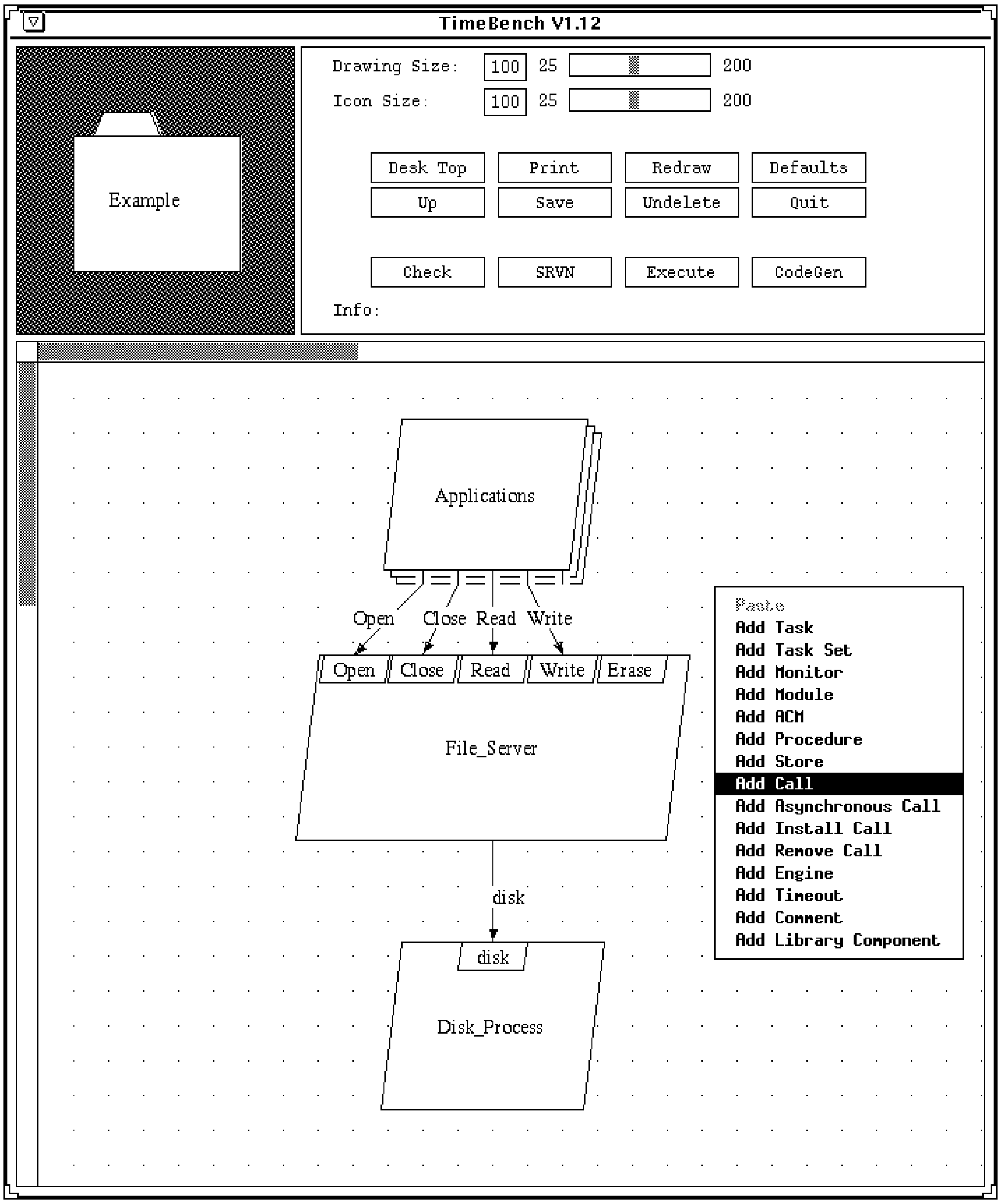

Figure 3: The TimeBench user interface showing the "Add Call" operation. The user is about to add an arc denoting a service request from the collection of tasks labeled *Applications* to the service entry labeled *Erase*.

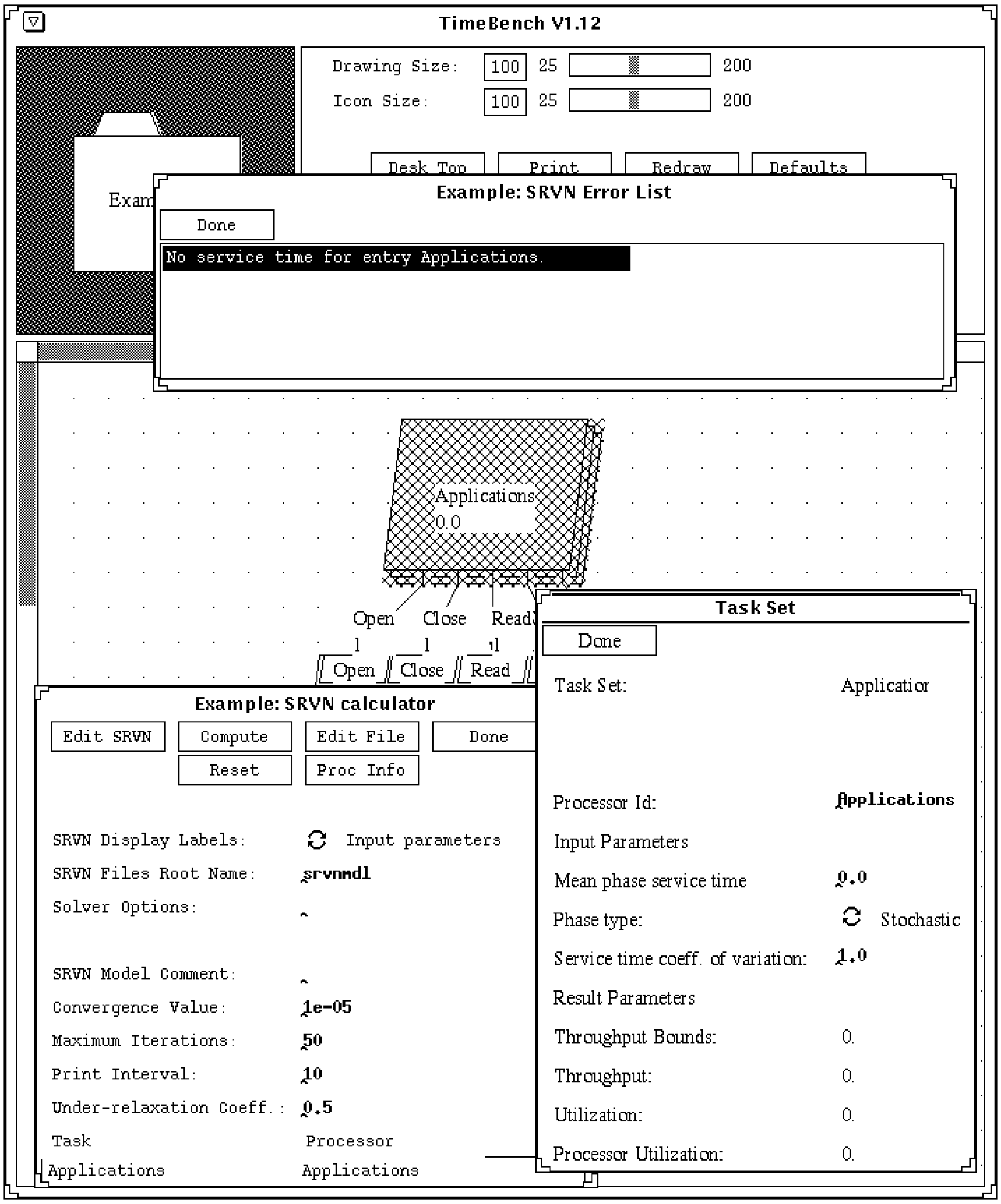

Figure 4: Entering SRVN data. The SRVN control panel is shown in the lower left corner of the tool. The error window is shown in the upper portion of the figure. The highlighted line in the error window has been used to open the SRVN data form for the task set *Applications*.

3.3 Performance Evaluation

The solution procedure is streamlined so only one command, "Solve", is used. The SRVN Calculator in TimeBench translates the data specified by the user within the design environment into a Stochastic Rendezvous Network performance model. The performance model associates all service time parameters with entries, so parameters specified for procedures and other passive entities in a MachineCharts design are aggregated by tracing the graph of procedure calls to give the total service time and visit ratios for the entry that executes them. After the selected solver is run, the results are displayed on the screen. They can be reviewed in three different ways:

1. On the model itself. By default, throughputs are displayed. Other performance results, such as utilization, can be displayed by changing the `SRVN Display Labels` item in the control panel.

2. For a particular node, by displaying its form.

3. Separately in a text window showing the output file generated by the solver.

4 Performance Evaluation Tools

The solvers for TimeBench performance models are implemented as separate programs which share a common input and output file structure. A new solver only has to conform to the input and output file formats, so solvers can be easily added or modified. Figure 5 shows the interactions between TimeBench and the solvers. Models are solved as follows:

1. TimeBench writes the performance model to a file.

2. The desired solver is invoked.

3. The performance results are read back into the tool and displayed.

The model and results files are retained so that they can be reviewed further or used as inputs to other programs.

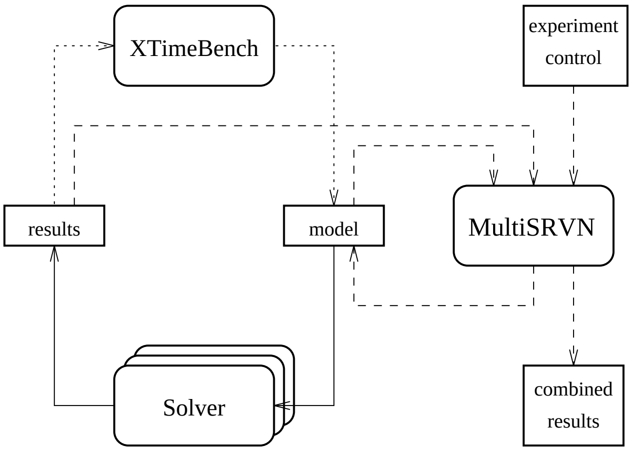

Figure 5: Tool Interaction

At present, TimeBench can call directly six different solution packages: three queueing-based solvers referred to as "SRVN6", "TDA" and "MOL", a Petri Net solver, a simulator named "ParaS-RVN", and a tool for computing throughput bounds. Table 2 lists some features that are useful in describing a SRVN model together with the extent to which each solver covers each feature. The features listed in the table are described in detail here. Beginning at the top, the scheduling discipline may be applied at the processors or in scheduling service to messages arriving at a task. In the latter case, priorities may be applied between requests for different service entries. Open arrivals come from outside the model to some entry in an assumed Poisson stream with a given rate. The phase type describes the model of control in each phase of service of an entry; a deterministic-type phase always has exactly the given number of requests, while a stochastic-type phase has a random number with the given mean. For deterministic phases, the coefficient of variation (the ratio of the variance over the square of the mean of the execution time distribution), C_v^2, can be varied. The default execution time distribution is exponential, giving a value for C_v^2 of one. Interprocessor communication delays are modeled as a given average delay with a value for each processor-pair. Asynchronous messages require no reply and do not block the sender. A "multi-server" refers to multiple copies of a server task that share a single queue of requesters. In an infinite server there is no upper limit on the number of copies. The "capacity" attribute refers to small, medium and large models, by which we mean about 5 tasks, about 25 tasks, or of unbounded size (for example, 1000 tasks).

Parameter	ParaSRVN	MOL	SRVN6	TDA	GSPN	Bounds
Scheduling†	FPHR	F(PHS)$^\diamond$	FPH	F	FR	F
Open arrivals	yes	no	yes	no	no	no
Phase type‡	SD	S	SD	S	S	SD
Vary C_v^2	yes	yes	yes	no	no	yes
Interprocessor delay	yes	no	yes	no	yes	no
Asynchronous sends	yes	no	yes	no	no	no
Multi-servers	yes	yes	no	no	yes	no
Infinite-servers	yes	yes	yes	no	no	yes
Capacity	large	large	medium	medium	small	large

† F: FIFO, P: Preemptive Priority, H: Head-of-Line Priority, R: Random, S: Processor Sharing
‡ S: Stochastic, D: Deterministic
$\diamond$ At Devices Only

Table 2: Solver Capabilities

The subsections that follow list briefly the techniques and advantages of each of the solvers and the bounds calculator. Lastly, we describe a separate tool called MultiSRVN which is used to conduct parametric runs using any of these solvers.

4.1 The ParaSRVN Simulation Solver

This solver has the fewest limitations, apart from the longer run-times needed for simulation. It supports all the features in the SRVN model and others. Task components require few resources so large models can be studied.

From a library of templates for tasks, processors, and queues, ParaSRVN creates instances of simulation objects for a discrete event-driven simulation. A simulation run collects standard statistics on throughputs, phase delays and queueing delays. Statistics are collected by block replications until throughputs have confidence intervals within a user provided tolerance, typically 1% of the mean.

The underlying simulation software is Neilson's PARASOL system [18], and is essentially a run-time kernel which runs lightweight threads representing simulated tasks. The threads are scheduled on simulated processors in simulated multiprocessor "nodes". The simulator maintains a simulated clock time which is advanced by special delay operations representing execution delays, and by the event scheduler. Task code is written in C or C++ and is linked to the run-time. PARASOL is a cross between a conventional discrete simulation library — which provides event handling, simulated time management, random variate generation, and statistics gathering functions, and a threads library — which provides thread management and coordination, and message passing primitives on user defined, multicomputer execution environments. PARASOL is used both as a simulation tool and for building distributed/parallel system emulators which can themselves be used as prototyping tools.

4.2 Heuristic Mean Value Analysis for Stochastic Rendezvous Networks — SRVN6

The method behind this solver has been described in detail in [33], with an earlier version in [32]. It considers the SRVN model one server at a time (including all of its entries), and estimates the delay to service requests arriving at the server (servers can be tasks or processors). Using special heuristics that are based on mean performance quantities and arrival instant probabilities, the delay is partitioned into components (for example, queueing delay). This gives a special-purpose Mean Value Analysis technique. The solution strategy within the submodel is iterative, related loosely to the Bard-Schweitzer approximate MVA algorithm for queueing networks [27]. The heuristics have been extended to include all the features of Table 1 except multi-servers. Errors in the reference task throughputs are normally within a few percent, although higher levels are occasionally observed. The strength of this model is its heuristics for a wide variety of situations.

This solver, and the MOL and TDA solvers that follow, consider submodels iteratively. The complexity of each iteration depends on the type of approximate MVA that is used. The iterations have polynomial complexity of a low order based on the number of entries or tasks in the model. As a result, they scale up to large systems much better than exact techniques.

4.3 Method of Layers (MOL)

The method of layers [24, 25] is also approximate MVA based, but with a solution approach and heuristics that differ from SRVN6. The approach views the performance model as a sequence of layers that are mapped onto a sequence of two level queueing network submodels. Performance estimates for the submodels are found using a modified version of the Linearizer algorithm for queueing networks [7]. The results are combined to provide performance estimates for the system as a whole. Clients are represented in the submodels using queueing network job classes. With this approach, clients can have high populations without a significant increase in the complexity or cost of performance evaluation.

Each task that provides service, and each device, is represented as a server in exactly one submodel. All the callers of the server are also represented in the same submodel, along with the callers' other servers. Thus when the Linearizer routine removes a customer from the queueing network, the joint impact of removing the customer on all of its servers and related clients is considered. This makes better use of information obtained by removing the customer than if each server is considered in a separate model. The combination of the use of Linearizer, which often provides superior accuracy when compared to Bard-Schweitzer, and the simultaneous solution of all servers in a layer, gives more accurate results than SRVN6 in some cases.

The method of layers also supports a heuristic that offers a limited form of synchronization between tasks. The feature is not supported by TimeBench, but has been used to model a class of generalized stochastic Petri Nets [23].

4.4 Task-Directed Aggregation (TDA)

The TDA algorithm was described in [20, 21]. It replaces most of the heuristic arrival-instant probability estimates in SRVN6 with estimates derived from a deeper theoretical analysis. Research on the Markov chain model for task interactions in a single-server submodel yielded an aggregated Markov chain that gives a set of equations for the arrival-instant probabilities. All but one are exact. Due to Markov chain aggregation, the complexity of the solution for the submodel was reduced from exponential to polynomial with respect to the number of clients and server entries [21]. The strength of the TDA algorithm is that it produces good results for multiple-entry servers with high service time variation across entries and large differences in client inter-arrival times. The results are an order of magnitude more accurate than SRVN6 in these cases, provided the server utilization is not too high.

A solver that combines the best features of SRVN6, MOL, and TDA is currently under development.

4.5 Markov Model Analysis via Stochastic Petri Nets

This solver translates the model into a generalized stochastic Petri Net (GSPN) in a format that is acceptable to the GreatSPN program of Chiola [8]. It then invokes the solver part of GreatSPN, which translates the model into a Markov Chain and solves it numerically. The TimeBench solver then reads the results file and imports the results for display.

This solver is "exact" (within a given tolerance for the numerical solution) and is useful for studying errors made by the approximate algorithms. However it suffers from state explosion which limits it to small systems, and our translator is also limited in the model features that it can handle, as seen in Table 2.

4.6 Throughput Bounds Solver

The previous solvers find point values for task throughputs, based on certain stochastic assumptions such as known distributions of service times and probability mechanisms for the requests. While these assumptions are often made in performance analysis they are usually quite hard to verify. The bounds solver makes minimal stochastic assumptions, and determines intervals which contain the values of throughput. The assumptions are that: requests are served in a FIFO manner at a server, task computation times during any phase are independent with any general Increasing Failure Rate distribution, and the number of rendezvous requests generated by tasks during any phase are independent random variables.

Three types of throughput bounds are considered [16]. A *no contention* upper bound is based on the optimistic supposition that a task never experiences any queuing delay when requesting service from servers. The *utilization-based* upper bound limits each server's utilization to one. The *lower bound* captures the pessimistic supposition that a reference task incurs the largest possible queueing delay at each server.

The upper and lower bounds on the throughput of each reference task give rise to a set of inequalities that are expressed in terms of the throughputs of other reference tasks. Because the throughputs of the tasks are unknown, it is difficult to extract the bounds in a closed form. A novel interval-arithmetic technique, supported by the BNR Prolog language [15], is used to compute numeric values for the bounds. A tool exists that takes as input an SRVN model (in the input file format shared by all of the solvers) and converts it into a BNR Prolog program. The execution of the program provides as output the bounds on reference task throughputs.

4.7 Multi-Point Solver for Sensitivity Analysis (MultiSRVN)

This is a facility for examining the effect of parameter variations. Although it is not integrated into TimeBench it is often used with the tool, to obtain insight into system behaviour under different performance scenarios.

Figure 5 shows how MultiSRVN fits into the tool set. TimeBench is used to create a model file. The user creates an experiment control file which defines which parameters are to be varied (one or two parameters), a set of values for each parameter, which solver to use, and which results are to be collected. This provides MultiSRVN with the information it needs to build a new model file for each parameter combination identified by the experiment control file. MultiSRVN invokes one of the above solvers to compute the results for each of the models, creating a set of corresponding result files. MultiSRVN then scans the result files, retrieves the results requested by the user, and produces tables or graphs as required. It can format the results as LaTeX tables, gnuplot or matlab data files and command scripts to plot graphics, as well as plain ASCII data files.

MultiSRVN helps to analyze trends in performance, locate important parameters (where small changes affect performance measures significantly), and play "what-if" scenarios. Whenever it is necessary to solve many similar models, MultiSRVN can be used to do the work automatically.

5 Example

This section describes the analysis of a small system which uses many of the salient features of the toolset. The example consists of a set of clients which make requests to a common application server and to a common file server. The file server is also used directly by the application server. All tasks run on their own processors.

Suppose that during the development process, it was determined that one of the system's functions could be performed either at the client tasks, or at the application server. The clients run input and screen management for the users, and can optionally also do a substantial amount of pre-processing of each request (checking, breakdown into file requests, etc.). Alternatively, the pre-processing could be executed by the application task, which would then continue to process the request and make the file accesses. However, the processors used by the Clients are significantly less powerful (say by a factor of ten) than the processor employed by the common application server. Both configurations are shown in Figure 6 complete with their parameters. Remember that before solving, the parameters of the pre-processing module are aggregated into the task which executes it. For instance, in the case of the "Server Compute" case, the phase execution times of the "Rqst" entry after aggregation are 1.4, 0.5, and 0. The pre-processing work is done by the module shown as a rectangle which is called from the Client task in the left-hand "Client Compute" case, and by the Application task in the "Server Compute" case.

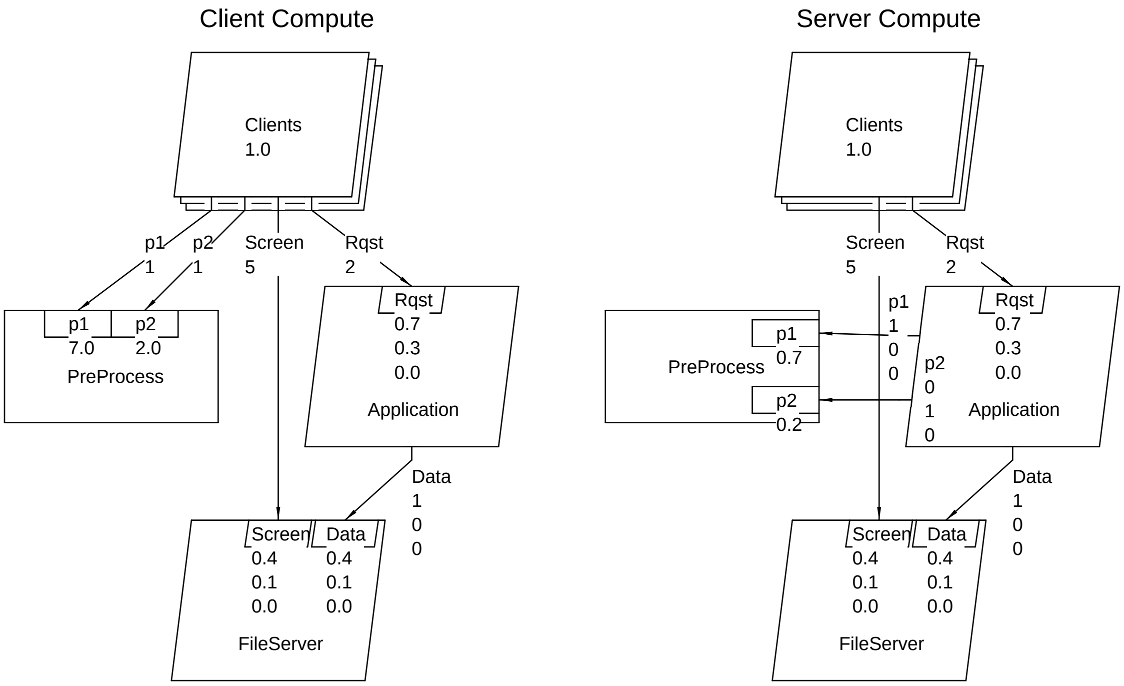

Figure 6: Example to study workload partitioning. The numbers under the entries are the mean phase service times and the numbers on the arcs are the mean visit ratios.

The parameters for both alternatives were used with a varying number of clients and solved using MultiSRVN and the MOL analytic solver. (The “Client Compute” case was also solved using the bounds solver; the results from this experiment are shown in Figure 8). The performance characteristics plotted in the graph in Figure 7 indicate clearly that the optimal placement of the pre-processing workload is dependent upon the number of clients, and that with more than three clients, the “Client Compute” alternative is better. This analysis also suggests that some intermediate partitioning of the pre-processing may be optimal for systems with three to six clients.

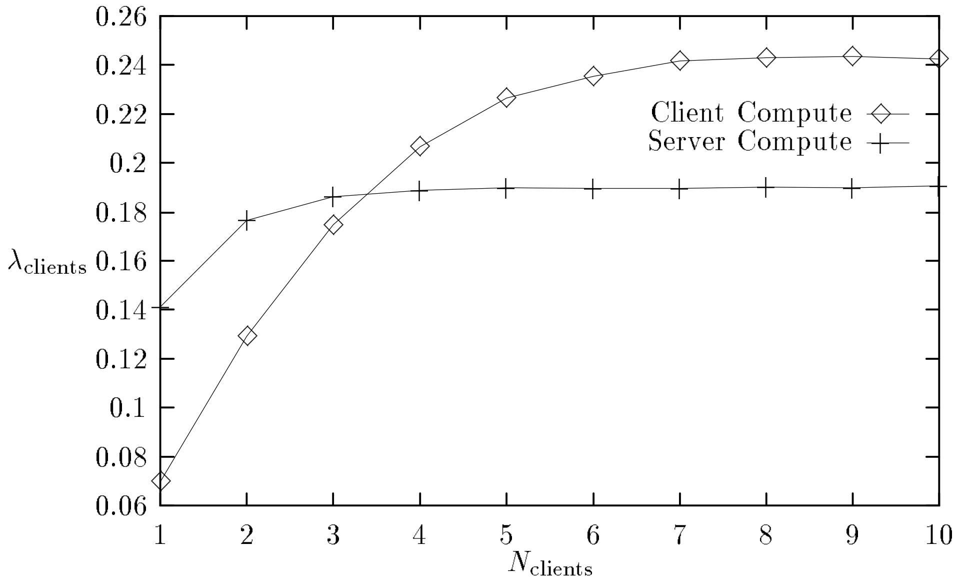

Figure 7: Comparison of throughput versus number of clients where functionality is moved between the clients and the server.

6 Conclusions

The TimeBench/SRVN tool set offers the possibility of early performance evaluation of designs for several kinds of layered service systems, including client-server, remote-procedure-call, and transaction processing systems. It has been used to model systems that include: an on-line transaction processing system with seven layers of servers and multiple tasks at each layer, a large system for capturing communications billing data, and a telecommunications service management architecture [14]. In these applications, the tools identified bottlenecks and performance tuning issues.

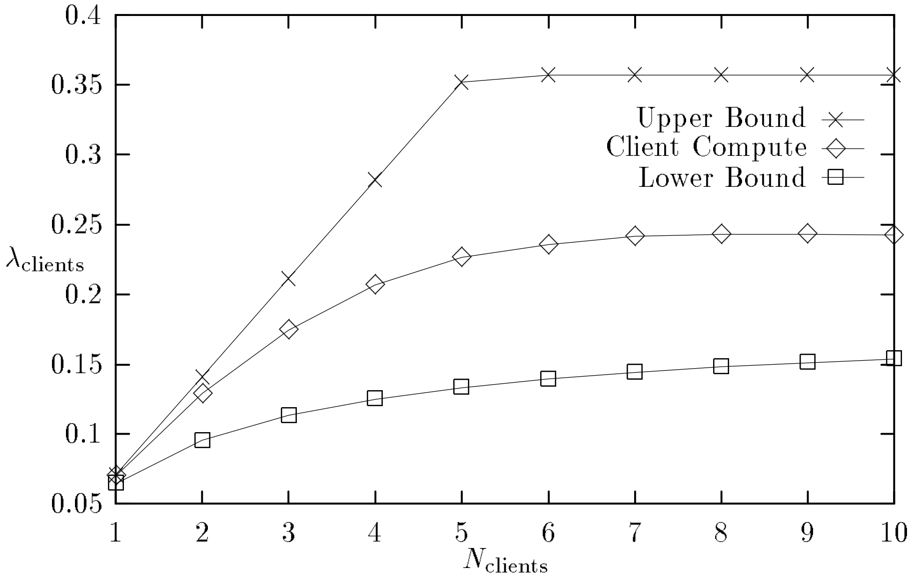

Figure 8: Upper and lower throughput bounds versus number of clients for the Client Compute test case.

Features that affect the performance of these systems include: the placement of modules in tasks, the population and threading levels of tasks, the allocation of tasks to processors, and the number of clients. These features can all be represented and manipulated using annotated MachineCharts. Performance estimates for alternative system designs can be found using our solvers. Relatively simple parameter entry, full parameter display, and parameter checking are features of the user interface.

Solutions of the performance model can be obtained by a simulation solver, or by one of three fast but approximate techniques adapted specially to this class of system. These approaches are complementary; analytic techniques can be used to explore alternatives, with simulation to confirm the results in the most important cases.

The fast solvers (SRVN6, MOL, TDA) are important for giving insight into how a design will behave once deployed. At present they give good accuracy (errors of a few percent, compared to simulation or to an exact solution of the same model) for systems which are layered and in which the service entries of a single task are not very different. Cases do occur with larger errors, such as 10% or more, and current research is aimed at identifying such cases and improving the approximations. These solvers are scalable, with low-order polynomial complexity. Ongoing research is also addressing multi-servers with different service entries, tasks which create parallel requests [22] and priorities [17].

A bounds solver based on minimal assumptions about the stochastic behaviour of the system is also included. A set of upper and lower bounds define a feasible throughput space which is guaranteed to contain task throughputs irrespective of service time distributions.

Acknowledgments

Ray Buhr, Ron Casselman and Gerald Karam were the key players in the genesis of TimeBench; the implementation was influenced by earlier work by Kevin Watson. We gratefully acknowledge the financial support of the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Telecommunications Research Institute of Ontario (TRIO).

References

[1] Andrew D. Birrell and Bruce Jay Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2(1):39–59, February 1984.

[2] Grady Booch. *Software Engineering with Ada.* Benjamin/Cummings, Menlo Park, CA, 2nd edition, 1987.

[3] Peter Buchholz. Hierarchies in colored GSPNs. In Marco Ajmone Marsan, editor, *Application and Theory of Petri Nets 1993*, number 691 in Lecture Notes in Computer Science, pages 106–125. Spriger-Verlag, Chicago, IL, June 1993.

[4] R. J. A. Buhr. *System Design with Ada.* Prentice Hall, Inc., Englewood Cliffs, New Jersey, 07632, 1984.

[5] R. J. A. Buhr. *Practical Visual Techniques in System Design; With Applications to Ada.* Prentice Hall, Englewood Cliffs, NJ, 1990.

[6] R. J. A. Buhr, G. M. Karam, C. M. Woodside, Ron Casselman, R. G. Franks, Hazel Scott, and Don Bailey. TimeBench: a CAD tool for real-time system design. In *Proceedings of the 2nd International Symposium on Environments and Tools for Ada (SETA2)*, Washington, D.C., January 1992. (Also as SCE tech report SCE-90-17).

[7] K. Mani Chandy and Doug Neuse. Linearizer: A heuristic algorithm for queueing network models of computing systems. *Communications of the ACM*, 25(2):126–134, February 1982.

[8] G. Chiola. A graphical Petri Net tool for performance analysis. In *Proceedings of the Third International Workshop on Modeling Techniques and Performance Evaluation*, France, May 1987. AFCET.

[9] *Introduction to OSF DCE.* Prentice Hall, 1992.

[10] Michael L. Fontenot. Software congestion, mobile servers, and the hyperbolic model. *IEEE Transactions on Software Engineering*, SE-15(8):947–962, August 1989.

[11] Giulana Franceschinis and Richard M. Muntz. Computing bounds for the performance indices of quasi-lumpable Stochastic Well-Formed Nets. In *5th International Workshop on Petri Nets and Performance Models*, pages 148–157, Toulouse, France, October 1993. I.E.E.E. Computer Society, A.C.M. Sigmetrics and AFCET.

[12] Oliver C. Ibe, Hoon Choi, and Kishor S. Trivedi. Performance evaluation of client-server systems. *IEEE Transactions on Parallel and Distributed Systems*, 4(1):1217–1229, November 1993.

[13] P. A. Jacobson and E. D. Lazowska. Analyzing queueing networks with simultaneous resource possession. *Communications of the ACM*, 25(2):142–151, February 1982.

[14] P. P. Jogalekar, G. Boersma, R. MacGillivray, and C. M. Woodside. TINA architectures and performance: A Telepresence case study. In *TINA 95*, Melbourne, Australia, February 1995. TINA Consortium. To Appear.

[15] Computing Research Laboratory. *BNR Prolog Reference Manual.* Bell Northern Research Ltd., Ottawa, Ontario, Canada, 1988.

[16] Shikharesh Majumdar, C. Murray Woodside, John E. Neilson, and Dorina C. Petriu. Performance bounds for concurrent software with rendezvous. *Performance Evaluation*, 13(4):207–236, 1991.

[17] J. W. Miernik, C. M. Woodside, J. E. Neilson, and D. C. Petriu. Performance of stochastic rendezvous networks with priority tasks. In T. Hasegawa, H. Takagi, and Y. Takahasi, editors, *Performace of Distributed and Parallel Systems*, pages 511–525. Elsevier Science, 1989.

[18] John E. Neilson. PARASOL: A simulator for distributed and/or parallel systems. Technical Report SCS TR-192, School of Computer Science, Carleton University, Ottawa, Ontario, Canada, May 1991.

[19] John E. Neilson, C. Murray Woodside, Dorina C. Petriu, and Shikharesh Majumdar. Software bottlenecking in client-server systems and rendezvous networks. Technical Report SCE-92-17, Department of Systems and Computer Engineering, Carleton University, Ottawa, Ontario, Canada, 1992. Submitted for publication to *IEEE Transactions on Software Engineering*, August 1992.

[20] Dorina C. Petriu. *Approximate Solution for Stochastic Rendezvous Networks by Markov Chain Task-Directed Aggregation.* PhD thesis, Carleton University, Ottawa, Ontario, Canada, 1991.

[21] Dorina C. Petriu. Approximate mean value analysis of client–server systems with multi-class requests. In *Proceedings of the 1994 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems.*, pages 77–86, Nashville, TN, U.S.A., May 1994. A.C.M. SIGMETRICS. *Performance Evaluation Review 22(1).*

[22] Dorina C. Petriu, Shikharesh Majumdar, Jing-Ping Lin, and Curtis Hrischuk. Analytic performance estimation of client-server systems with multi-threaded clients. In *Proceedings of the International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'94)*, pages 96–100, Durham, NC, USA, January 1994.

[23] Jerome A. Rolia and Kenneth C. Sevcik. Fast performance estimates for a class of generalized stochastic Petri nets. In *Computer Performance Evaluation '92: Modelling Techniques and Tools*, pages 21–33. Edinburgh University Press, August 1993.

[24] Jerome Alexander Rolia. Performance estimates for systems with software servers: The lazy boss method. In Ignacio Casas, editor, *VIII SCCC International Conference On Computer Science*, pages 25–43, Santiago, Chile, July 1988. Chilean Computer Science Society.

[25] Jerome Alexander Rolia. Predicting the performance of software systems. Technical Report CSRI-260, Computer Systems Research Institute, University of Toronto, Toronto, Canada, January 1992.

[26] C. H. Sauer, E. A. MacNair, and J. F. Kurose. The research queueing package: Past, present and future. In *Proceedings of the National Computer Conference*, pages 273–280, Houston, TX, June 1982.

[27] P. Schweitzer. Approximate analysis of multiclass closed networks of queues. In *Proceedings of the International Conference on Stochastic Control and Optimization*, Amsterdam, 1979.

[28] Scientific and Engineering Software, Inc., 4301 Westbank Drive, Building A., Austin Texas. *SES/Workbench*, 2.1 edition, February 1992.

[29] Connie U. Smith. Applying synthesis principles to create responsive software systems. *IEEE Transactions on Software Engineering*, SE-14(10):1394–1408, October 1988.

[30] Connie U. Smith. *Performance Engineering of Software Systems.* The SEI Series in Software Engineering. Addison-Wesley, 1990.

[31] Vidar Vetland, Peter Hughes, and Arne Sølvberg. Improved parameter capture for simulation based on composite work models of software. In *Proceedings of the 1993 Summer Computer Simulation Conference*, Boston, MA, July 1993.

[32] C. Murray Woodside. Throughput calculation for basic stochastic rendezvous networks. *Performance Evaluation*, 9:143–160, 1989.

[33] C. Murray Woodside, John E. Neilson, Dorina C. Petriu, and Shikharesh Majumdar. The stochastic rendezvous network model for performance of synchronous multi-tasking distributed software. *IEEE Transactions on Software Engineering*, 1994. Accepted for publication.