1981

Metamodeling and its Application to Queueing Networks

Jeffrey P. Buzen

Subhash C. Agrawal

Report Number:
81-385

Metamodeling and Its Application to Queueing Networks

Jeffrey P. Buzen

BGS Systems, Inc.
1 University Office Park,
Waltham, MA. 02254;

Subhash C. Agrawal

Purdue University,
Department of Computer Sciences,
West Lafayette, IN. 47907.

CSD-TR-385

ABSTRACT

A framework is presented for characterizing and analyzing relationships among alternative mathematical models of a single underlying system. This framework is intended to facilitate the comparative analysis of modeling strategies and the development of new techniques for exact and approximate solution of complex models. A related objective is to facilitate communication among researchers, practitioners and students of modeling.

The concept of a state space transformation, which is a central component of the metamodeling framework, is introduced to provide a mechanism for expressing the way one model can be mapped into another. After discussing state space transformations in general terms, the mathematical properties of several specific transformations are examined. Applications of metamodeling to the shadow CPU technique and the aggregate server method are then presented.

Key Words and Phrases: aggregate server method, approximation, metamodeling, operational analysis, performance evaluation, preemptive priority, product form, queueing networks, serialization delays, shadow CPU algorithm, state space, transformations.

Categories and Subject Descriptors: C.4.8 Performance (Operational analysis, Queueing theory)

This work was supported, in part, by NSF grant MCS76-01729. During the final stages of the preparation of the paper, Subhash Agrawal was supported by an IBM Graduate Research Fellowship.

Metamodeling and Its Application to Queueing Networks

1. Introduction

Over the past two decades, a succession of increasingly complex mathematical models have been used to analyze computer performance. The first models treated isolated elements within a system such as individual CPU's [16], disks [11], and drums [14]. In the early 1970's, the development of queueing network models made it possible to represent all the principal elements of an entire system in a single model [7, 19]. Queueing network models have been extended considerably since their inception, and they are now able to deal, exactly or approximately, with refinements such as multiple job classes [4], priority scheduling [22], and queueing for special passive resources such as main memory [5, 10], I/O path elements [3], data base granules [20], and critical sections [1].

One noteworthy trend that has emerged during this evolution is the use of solution procedures that are based on integrating or reconciling several alternate models of a single physical system. The earliest examples of this approach involved reasonably straightforward integration of hierarchically nested flow-equivalent sub-models [5]. More recent examples involve iterating among partially overlapping, non-nested models until they converge to the desired solution [15].

In such cases, the formal mathematical structure of each sub-model can be specified precisely using either stochastic modeling [17, 18] or operational analysis [9, 13]. There is, however, presently no general framework for expressing the way different sub-models relate to each other and to the underlying system they represent. The lack of such a framework makes it difficult to evaluate and compare alternative modeling procedures, hampers communication among researchers, practitioners and students, and – perhaps most significantly -- limits the complexity of the modeling procedures that can be developed by constraining researchers to work in an ad hoc manner.

Metamodeling, as introduced in this paper, attempts to resolve these problems by providing a conceptual framework for dealing with models and the modeling process. This framework makes it possible to identify transformations that map one model into another, and to evaluate the impact of these transformations on various quantities of associated with each model. Since a number of different transformations may be applied successively during the analysis process, the ability to identify and analyze each transformation can be of significant value in conceptualizing the modeling and solution process, in partitioning these processes into smaller steps that are easier to deal with, and in understanding the structure of the model and communicating it to others.

2. Overview of the Paper

The first step in developing a metamodeling framework is to identify the principal components of the class of models that are being considered. This paper focuses on queueing network models and, to a limited extent, on their interpretation under operational analysis [9, 13]. Metamodeling frameworks could, in principle, be developed for other categories of models, but such extensions are beyond the scope of this paper.

Section 3 briefly reviews some of the most important components of queueing network models such as customer classes, queues, and state space representations. These concepts apply equally well to both operational and stochastic interpretations of queueing network models. Section 4, which discusses behavior sequences and their associated variables, is more operationally oriented. Some standard definitions for the operational variables associated with multi-class queueing network models appear in this section. These definitions are essential for the mathematical treatment of state

space transformations presented in subsequent sections.

The concept of a state space transformation is formally introduced in Section 5. A number of specific transformations are identified and their impact on some of the operational variables discussed in Section 4 is analyzed. The discussion of state space transformations is carried out in the context of multi-class queueing network models, but extensions of these concepts to other types of models should present no major difficulties.

Section 6 shows how certain iterative solution techniques are related to state space transformations and their inverses. Many of the most important approximations used in the solution of queueing network models are based on such iterations.

The major concepts presented in Section 3 through 6 are brought together in a comprehensive example in Section 7. The example focuses on the aggregate server method as presented by [1]. The principal state space transformations associated with this technique are identified, their mathematical properties are reviewed, and the role of the iterative technique is clarified. It is expected that other approximation techniques could be treated in a similar manner. Section 8 concludes the paper by stating some of the more important applications and implications of the ideas presented here.

3. State Spaces and Other Model Components

Queueing network models are traditionally described by identifying their customer classes, servers and queues. Other components that are also identified include the service disciplines at each server, the nature of the arrival and service processes, and the presence of the special constraints associated with passive resource contention, blocking and similar factors.

One aspect of a queueing network model that is usually not identified explicitly is the state space. However, the state space is one of the most critical components to consider when discussing relationships among different models. For this reason, the nature of state spaces will now be examined in some detail.

Consider the operation of a queueing network model over an interval of time. Assume some observer records certain information concerning the model - the state of the model - at each instant. For example, the observer may choose to record the number of customers queued at a server, the amount of service already received by the customer in service, or the amount of waiting time accumulated by each customer in a queue.

For any such observer, the set of all possible values these observations can take will be referred to as a *state space*. Note that the structure of a model's state space is closely related to other components of the model. For example, in a multi-class product form queueing network with R customer classes and K servers, the set of information usually recorded is the number of customers of each class present at each server. In this case, the state space has the form $(n_{11}, \cdots, n_{\tau k}, \cdots, n_{RK})$ where $n_{\tau k}$ is the number of customers of class τ at server k. Given a state space of this form, the number of customer classes and the number of servers is immediately apparent.

Because of the close relationship between a state space and the other components of a model, relationships among models can be precisely described by identifying the transformations that map one state into another. The concept of using state space transformations to characterize and analyze relationships among models represents one of the most important aspects of metamodeling as presented in this paper. The transformations are discussed in Section 5.

4. Behavior Sequences and Derived Variables

Each observation of a queueing system can be regarded as a single point in the associated state space. A time ordered sequence of such points constitutes a record of the system's observed behavior during an interval of time. The term *behavior sequence* is widely used in operational analysis [9, 13] to refer to such a record.

Given a particular behavior sequence and its associated state space, it is possible to define a number of operational variables that reflect relevant properties of the system. For example, consider the case where the system is a single server queue and the behavior sequence is a function $N(t)$, the number of customers at the server, $0 \le t \le T$.

In this case following operational variables can be defined directly in terms of $N(t)$ using standard operational definitions [13].

$T(n)$ = Total time during which $N(t)=n$.

$A(n)$ = Number of customers which upon arrival find
n customers already in the system, i.e., number of arrivals during $T(n)$.

$C(n)$ = Number of customers which upon departure leave $n-1$ customers behind.

Then, define derived variables:

$p(n)$ = Proportion of time that there are n customers at the server, i.e., $T(n)/T$.

$\alpha(n)$ = arrival rate function, $A(n)/T(n)$.

$s(n)$ = service time function, $T(n)/C(n)$

s = mean service time, $(T-T(0))/\sum_{n>0} C(n)$.

W = accumulated waiting time,* $\sum_{n>0} nT(n)$

R = mean response time, $W/\sum_{n>0} C(n)$.

We can similarly compute the relevant statistics for a multi-class model. For example, in a multi-class, single server model, assume the state vector is described as $\underline{n}(t)=(n_1(t), \cdots, n_R(t))$, where $n_r(t), r=1, \cdots, R$ is the number of class r customers at the server at time t. Basic operational variables can be defined as follows:

$T(\underline{n})$ = Total time during which $n_r(t)=n_r$, $r=1, \cdots, R$

$A_r(\underline{n})$ =number of class r customers which upon arrival find
$\underline{n}$ (vector of) customers ahead of them
= number of class r arrivals during $T(\underline{n})$

$C_r(\underline{n})$ = number of class r completions that leave $\underline{n}-\underline{1}_r$ customers behind

Now we can define derived variables as shown below:

$T_r(n)$ = Total time during which n class r customers are present at the server
$= \sum_{n_r=n} T(\underline{n})$
$= \sum_{n_r=n} T(\underline{n})$

$C_r(n)$ = number of class r completions which leave $n-1$ class r customers behind
$= \sum_{n_r=n} C_r(\underline{n})$

* Also note that if we have a queue without a server, we compute accumulated waiting time but not service time.

$$p(\underline{n}) = T(\underline{n})/\,T$$
$$p_r(n) = \sum_{n_r=n} p(\underline{n})$$
$$\approx T_r(n)/\,T$$

$S_r(\underline{n})$ = service function for class r customers as a function of network population
$= T(\underline{n})/\,C_r(\underline{n})$
. . .

When computing certain derived variables from a behavior sequence, it is sometimes necessary to have auxiliary information such as whether a service discipline is processor sharing or first come, first served, or whether a server has a queue dependent processing rate. The interpretation and importance of this structural information is discussed further in [2].

5. State Space Transformations

Now that the concepts of state space, behavior sequence and derived variables have been identified, it is possible to discuss state space transformations. Recall that the motivation for metamodeling is the desire to create a conceptual framework for expressing and analyzing the way different models of the same underlying system relate to each other.

The approach we will take is based on the idea of identifying the state space of each model and then specifying transformations between state spaces as mathematical functions. Once these functions are identified, one can express the functional relationships among the derived variables of the associated models. Some examples of useful state space transformations which have been implicitly employed by queueing network researchers are presented below. In all cases, the transformations will map a primary model into a secondary model. The primary model will, in general, contain more detailed information, but the transformations will not always involve loss of information.

5.1. Load Concealment Transformation

The load concealment transformation operates on an individual server in a state space. In the primary model, the customers at that server are divided into several classes. In the secondary model, some of the classes are concealed: they are still present in the underlying system, but they are not reported in secondary model.

The other classes which are explicitly represented will be called visible classes. Under this transformation, service time that is actually due to the concealed classes, and that takes place when customers from the visible classes are present, is now attributed to the visible classes. This affects certain derived variables such as the time per visit for the visible classes.

Suppose, for example, there are R customer classes in the primary model and one (say, class 1) in the secondary. Assume the transformation is being applied to server i which uses processor sharing service discipline. Then the transformation is defined as follows:

In the primary model

$n_{ri}(t)$= the number of customers of class r at server i at time $t\,(j=1, \cdots, r)$

In the secondary model where customers of classes 2, $\cdots$, R are concealed

$n_i(t)$= the total number of visible (class 1) customers at server i at time t

Then a load concealment transformation that conceals classes 2, $\cdots$, R can be expressed simply as

$$n_i'(t)=n_{1i}(t)$$

The implications of this transformation become apparent when computing derived quantities. Let $T_i'(n)$ be the amount of time there are n visible (class 1) customers at server i in the secondary model. Also, let $T_i(\underline{n_i})$ be the amount of time there are n_{ri} class r customers at server i in the primary model. Then

$$T_i'(n)=\sum_{n_{1i}=n} T_i(\underline{n_i})$$

Number of visible (class 1) customer completions can be simply given as

$$C_i'(n)=C_{1i}(n)$$

Above $C_{ri}(n)$ is a logical extension of $C_r(n)$ given for the single server/queue system in section 4. Other operational quantities can be defined analogously. It then follows that

$$p_i'(n)=\sum_{n_{1i}=n} p_i(\underline{n_i})$$

and

$$S_i'(n)=T_i'(n)/\,C_i'(n)$$

Therefore, the service demand per visit for class r customer is

$$D_{ri}=\frac{\sum_{n_{ri}\geq 1}\dfrac{n_{ri}}{n_{1i}+\cdots+n_{Ri}}T_i(\underline{n_{ri}})}{\sum_{n\geq 1}C_{ri}(n)}$$

The contention due to the presence of unvisible customers elongates the apparent demand of the visible customers. This apparent, stretched out service demand is then given as

$$D_i'=\frac{\sum_{n\geq 1}T_i'(n)}{\sum_{n\geq 1}C_i'(n)}$$

Therefore, the service time adjustment factor, H_{1i}, which is the reciprocal of service time elongation factor is given by

$$\begin{aligned}H_{1i}&=\frac{D_{1i}}{D_i'}\\&=\frac{\sum_{n_{1i}\geq 1}\dfrac{n_{1i}}{n_{1i}+\cdots+n_{Ri}}T_i(\underline{n_i})}{\sum_{n\geq 1}T_i'(n)}\\&=\frac{\sum_{n_{1i}\geq 1}\dfrac{n_{1i}}{n_{1i}+\cdots+n_{Ri}}p_i(\underline{n_i})}{\sum_{n\geq 1}p_i'(n)}\\&=\frac{\sum_{n_{1i}\geq 1}\dfrac{n_{1i}}{n_{1i}+\cdots+n_{Ri}}p_i(\underline{n_i})}{p_i'(n\geq 1)}\end{aligned}$$

$p_i'(n\geq 1)$ is simply the probability that there is at least 1 class 1 customer present at the server i (in the original model, and therefore, in the secondary model).

As another example of the load concealment transformation, consider Sevcik's shadow server technique for analyzing preemptive priority queues [22]. In this technique, a single server with n priority classes is transformed into n separate servers, each serving customers of a single priority class. This can be viewed as the result of n load concealment transformations, where each transformation conceals customers of all priority classes except the one being considered. For $n=2$, with H denoting the high priority class, and L denoting the low priority class, it is easy to see that intrinsic service demands are

$$D_H = \frac{T(n_H \geq 1, n_L \geq 0)}{C_H}$$

and

$$D_L = \frac{T(n_H = 0, n_L \geq 1)}{C_L}$$

where C_H and C_L are the number of class H and class L completions respectively. The apparent service times of the transformed servers are

$$D_H' = \frac{T(n_H \geq 1)}{C_H}$$

and

$$D_L' = \frac{T(n_L \geq 1)}{C_L}$$

Thus, it is easy to see that the service time elongation factors are

$$F_H = \frac{D_H'}{D_H} = 1,$$

and

$$\begin{aligned} F_L &= \frac{D_L'}{D_L} \\ &= \frac{T(n_L \geq 1)}{T(n_H = 0, n_L \geq 1)} \\ &= \frac{p(n_L \geq 1)}{p(n_H = 0, n_L \geq 1)} \\ &= \frac{p(n_L \geq 1)}{p(n_H \geq 0, n_L \geq 1) - p(n_H > 0, n_L \geq 1)} \\ &= \frac{p(n_L \geq 1)}{p(n_L \geq 1) - p(n_H > 0, n_L \geq 1)} \end{aligned}$$

Now if we *assume* that

$$p(n_H > 0, n_L \geq 1) = p(n_H > 0) \cdot p(n_L \geq 1),$$

then we get

$$F_L = \frac{1}{1 - p(n_H > 0)} = \frac{1}{1 - U_H}.$$

This is the equation that was proposed by Sevcik. Note that this careful analysis has revealed an approximation that Sevcik had implicitly made and thus illustrates one of the benefits that are reaped by using this approach.

Also note that in the two examples presented above we have obtained two different sets of equations expressing the relationship between the primary and secondary models even though the same name is given to the transformation. It is to illustrate the *conceptual similarity* of the two analyses, namely, concealment of "undesirable" loads. By understanding such conceptual similarities it will be possible to identify

situation where a known transformation may be applicable and thus model development time can be reduced.

5.2. Class Aggregation Transformation

In this very common transformation, two or more classes in a multiple class model are aggregated into a single class in the secondary model. For example, if classes 1, 2 and 3 at server i are aggregated into a single class, the transformation maps $n_{1i}(t)$, $n_{2i}(t)$, and $n_{3i}(t)$ in the primary model into a single $n_i(t) = n_{1i}(t) + n_{2i}(t) + n_{3i}(t)$ in the secondary model. The relationships among associated operational variables and derived quantities follow immediately.

5.3. Server Aggregation Transformation

Server aggregation transformation is another very important transformation. This transformation translates a multi-server queueing network into a single server. In its general form, it is same as "garden-variety" decomposition [5, 10, 12, 13], and is most useful when it is applied to a single class queueing (sub) network. This transformation is exact for product form queueing network.

A special case of this transformation in which number of customers in the subnetwork is limited to 1 is of special importance. The resulting server is load independent, and the service time of customers at this equivalent server is simply the sum of the service times at the individual servers in the subnetwork.

This transformation is used best when there exists a constraint on the maximum number of customers in the subnetwork, and when this limit is reached, an arriving customer has to wait in a queue for entry into the subnetwork. When a customer departs from the subnetwork, a customer from this queue is allowed to enter in the subnetwork. Owing to these considerations, this transformation is usually followed by server/queue concatenation transformation described below.

5.4. Server/Queue Concatenation Transformation

Rationale for this transformation has been described in the previous subsection. In the primary model, there is a server-queue pair such that the server and the queue are represented separately in the state space, and when a customer arrives at this pair, it first enters the queue, and if the concurrency limitations permit, it immediately proceeds to the server. Otherwise, it awaits its turn in the queue.

In the secondary model, this is represented as a single server (as in a conventional queueing network), and the queue is eliminated from the network. All customers present at the queue are now considered to be at the server explicitly.

6. Iterations

In some of the transformations just discussed, it is necessary to know certain state probabilities and performance measures in the primary model in order to calculate other state probabilities and performance measures in the secondary model. If the required quantities cannot be obtained from some other source such as available data or the solution of another model, it is sometimes possible to compute these quantities using an iterative approach.

These iterations generally proceed as follows. Begin with an initial estimate of the required quantities in the primary model. Using this estimate, carry out the transformation, solve the transformed model, and then apply the inverse of the transformation to obtain a new estimate of what the quantities in the primary model should have been. Using this new estimate, apply the transformation a second time and again solve the secondary model. The inverse transformations can now be applied and another cycle in the iteration can be carried out. The procedure is repeated until there is no

significant change in the values of the required quantities from one iteration to the next. Section 7 provides a detailed example of such an iterative procedure is applied to the aggregate server method.

The very same approach is applicable to the case where the iteration is between two or more models of the same system [15]. In this case, one model provides the estimates required for constructing the second model, and the second model provides the required measures for the first.

7. Aggregate Server Method for Analyzing Serialization Delays

In this section we will present the aggregate server method for analyzing serialization delays in computer systems [1] in the general metamodeling framework discussed above. However, before discussing the details, we will provide an overview of the aggregate server technique. [1]

7.1. Aggregate Server Method: An Overview

The aggregate server method is an approximate, iterative technique for analyzing the delays programs encounter while waiting for entry into critical sections, nonreentrant subroutines, and similar software structures that cause processing to become serialized. The method employs a conventional product form queueing network comprised of servers that represent actual I/O devices and processors, plus additional aggregate servers that represent serialized processing activity. The parameters of the product form network are adjusted iteratively to account for contention among serialized and non-serialized customers at each physical device.

The basic idea behind the aggregate server method is quite simple, and generalizes directly from consideration of the restricted case where there is a single customer class and a single critical section (one serialized phase). Consider such a network containing N customers and K servers. Let D_i be the total service time per job at server i.

Note that each D_i can be regarded as consisting of two components: D_{0i}, which is the total service time per job at server i that occurs outside the critical section, and D_{1i}, which is the total service time per job at server i that occurs inside the critical section. The aggregate server technique is based on the idea of adding an additional (aggregate) server to the network to represent the serialized processing in the critical section, and then regarding the expanded $K+1$ server network as having a conventional product form solution.

Let Y_i for $i=1,2,\cdots,K+1$ represent the total service time per job at server i in the expanded $K+1$ server method. The key step in the aggregate server method is determining appropriate values for the Y_i as a function of the D_{ji} and the number of customers N. These service times are expressed as $Y_i=D_{0i}/H_{0i}$ for $i=1,\cdots,K$ and $Y_{K+1}=D_{11}/H_{11}+\cdots+D_{1K}/H_{1K}$, where H_{0i} and H_{1i} are service time adjustment factors for nonserialized and serialized processing respectively. For given phase of processing they reflect the amount of contention due to processing in other phases at physical device i. Or equivalently, H_{ji} is the fraction of device i's capacity that is available for processing customers in phase j.

Below we provide an outline of the algorithm for 1 critical section. Complete algorithm is given in [1].

---

1. To provide a historical perspective, we note that the concept of metamodeling grew out of the efforts to develop and understand the aggregate server method in very simple terms. When viewed in the metamodeling framework, specially after foregoing the mathematical notation for the time being, it is much easier to understand the aggregate server method and other approximation techniques.

Step 0: Assume $H_{ji}=1$. (It corresponds to ignoring contention among customers in different phases of processing.)

Step 1: Construct the expanded network with $K+1$ servers and a single customer class.

Step 2: Solve the expanded network.

Step 3: The product form solution yields the joint distribution of customers in the expanded network. Using the current values in of H_{ji}, map this network into the original non-product form network with K physical devices and two phases. From this mapping, estimate the queue length distributions of each of the K physical devices in the non-product form network. These distribution provide information about the amount of contention that occurs among customers of different phases. Using this information, compute new estimates of the H_{ji}'s.

Step 4: If new H_{ji}'s differs from old H_{ji}'s significantly, then return to step 1 with new H_{ji}'s, otherwise compute performance measures and STOP.

7.2. Aggregate Server Method: A Metamodeling Perspective

Following discussion is in the framework of metamodeling. We hope to illustrate how powerful abstraction and idea of transformations can help in communicating ideas and algorithms between analysts, also provide insights into development of new algorithms for solving queueing networks.

The aggregate server technique works by applying a series of transformations on the primary model of a computer system. Basically, from the primary model of the system, we construct $Z+1$ similar networks - one for each phase of processing, one non-serialized and Z serialized. The servers in each of these networks are degraded to reflect the contention due to processing in other phases, and the amount of degradation is iteratively calculated. Depending on the current phase of processing, the customer is assigned to the network for the phase. Since only one customer can be inside a serialized phase at any time, we are able to collapse all the whole network into an equivalent server, leading to $K+Z$ server aggregate server network. We will now present the primary model of the system and successive application of various transformations discussed earlier to obtain the aggregate server network. First we summarize the notation in Table 1.

1. Underlying System's "Natural" Model

The "natural" model of a computer system consists of a number of servers at which jobs are waiting for or receiving service and some logical queues corresponding to serialized phases where jobs wait for their turn to begin processing in serialized mode. At this level, the state-space of the system can be described as $(\underline{n}_1, \underline{n}_2, \cdots, \underline{n}_K; r_1, r_2, \cdots, r_Z)$ where: $\underline{n}_i$ is the population vector $(\overline{n}_{0i}, \overline{n}_{1i}, \cdots, \overline{n}_{Zi})$ at server i, n_{zi} is the number of jobs in phase z ($z=0$ means unserialized; and $z \geq 1$ means serialized phase z and $n_{zi} \in \{0,1\}$) at the server i; r_z is simply the number of customers waiting to enter the z^{th} serialized phase of processing. (Note that due to processor sharing service discipline assumption, only the number of customers in each phase is important.)

2. Load Concealment Transformation

First we apply the load concealment transformation to each server in this model to produce a separate slowed down server for each phase. The transform of server i for phase z processing, indexed as zi will process customers in phase z of processing only. (Note that here the phase of processing a customer is in denotes its current class.)

Table 1: Notation

Symbol	Definition
J	set of jobs (customers) observed in the network
j	index of a job
R	number of job classes
r	a job class index
P	set of properties or attributes
$P_j(t)$	set of attributes or properties of customer j at time t
C	number of completions
A	number of arrivals
p	probability of some event happening
N	total number of customers in the network (i.e., MPL)
K	number of servers in the network
Z	number of serialized phases
i	index of servers under consideration $i=1,...,K$ original devices $i=K+1,...,K+Z$ aggregate serialized servers
z	phase index $z=0$ nonserialized phase $z=1,...,Z$ serialized phase
Z_i	number of serialized phases in which a customer may visit device i, $i=1,...,K$
n_{zi}	number of customers at device i while visiting phase z. (n_{0i} is the number of nonserialized customers receiving service at device i, and if $z>0$, $n_{zi} \in \{0,1\}$)
$\underline{n_i}$	vector $(n_{0i}, n_{1i}, \cdots, n_{Zi})$, representing the number of customers per phase at device i
m_z	number of customers inside serialized phase z, $z \geq 1$, i.e., $m_z = \sum_{i=1}^{K} n_{zi}$. $m_z \in \{0,1\}$.
τ_z	number of customers waiting for entry into the serialized phase z, $z \geq 1$.
n_z	number of customers at serialized phase z. $n_z = \tau_z + m_z$.
D_{zi}	total service time requirement (demand) at server i while in phase z
H_{zi}	service time adjustment factor for a phase z customer at device i
Y_{zi}	stretched out service time requirement at server i for phase z, i.e., service time modified to reflect the effect of contention due to customers in other phases $Y_{zi} = D_{zi} / H_{zi}$
Y_i	service time requirement at server i in the aggregate server network: $$Y_i = \begin{cases} Y_{0i} & i=1,...,K \\ \sum_{l=1}^{K} Y_{i-K,l} & i=K+1,...,K+Z \end{cases}$$

These $(Z+1)*K$ servers can be considered to represent $Z+1$ isomorphic networks. The z^{th} ($z = 0, 1, \cdots, Z$) isomorphic network consists of servers with indices zi, $i = 1, 2, \cdots, K$ and is visited by only phase z customers. The one corresponding to $z=0$ is for non-serialized processing. The remaining Z networks correspond to the Z serialized phases.

The state vector is now represented as $((n_{zi}, i = 1, 2, \cdots, K), z = 0, 1, \cdots, Z; r_z, z = 1, 2, \cdots, Z)$. Each of the populations n_{zi} and r_z are directly derived from the original state representation. Note that only the order of various terms has been changed.

The service time requirement of class z customer at server zi is related to the service time requirement at server i in the natural model by service time adjustment factor H_{zi}. I.e., knowing H_{ji}'s, we can compute service time parameters for the transformed model. Following relations, (equations 3 and 4 of Agrawal and Buzen [1]), can be used to compute the necessary service time adjustment factors:

For non-serialized phase ($z=0$):

$$H_{0i} = \frac{\sum_{r=1}^{N} \sum_{k=0}^{\min(Z,N-r)} \frac{r}{r+k} p(n_{0i}=r \wedge \sum_{q=1}^{Z} n_{qi}=k)}{p(n_{0i} \geq 1)}$$

For serialized phase z, $z \geq 1$:

$$H_{zi} = \frac{\sum_{k=1}^{\min(Z,N)} \sum_{r=0}^{N-k} \frac{1}{r+k} p(n_{zi}=1 \wedge n_{0i}=r \wedge \sum_{q=1}^{Z} n_{qi}=k)}{p(n_{zi} \geq 1)}$$

3. Server Aggregation Transformation

We can aggregate servers in each of the isomorphic networks corresponding to a serialized phase ($z \geq 1$) into a single aggregate serialized phase server to represent the serialized processing. In our case, due to concurrency limitation, only one customer can be receiving service at these servers. Therefore, we can aggregate these servers (zi,~ $i = 1, 2, \cdots, K$) into corresponding (z^{th}) serialized phase server (with index $z+K$) by simply summing up the service requirements at the servers zi. The state vector now becomes $(n_{0i}, i = 1, 2, \cdots, K; m_z, z = 1, 2, \cdots, Z; r_z, z = 1, 2, \cdots, Z)$, where $m_z = \sum_{i=1}^{K} n_{zi}$. State probabilities can be obtained by summing up appropriate state probabilities from step 2.

4. Server-Queue Concatenation Transformation

We still have the logical serialized phase queues separated from the serialized phase servers constructed in the step 3 outlined above. We now concatenate the two together to show that the customers wait for entry into a serialized phase. At this stage, state vector is represented as $(n_{0i}, i = 1, 2, \cdots, K; n_z, z = 1, 2, \cdots, Z)$, where $n_z = r_z + m_z$ in the step 3. State probabilities at this stage can be obtained, trivially, from the ones available at step 3 above.

5. Obtaining the Performance Measures

The resulting network produced by applying the transformations mentioned above may not satisfy the homogeneity assumptions. However at this stage the principal reasons for the original inhomogeneity have been accounted for by the transformations, and so we now assume that the network satisfies the homogeneity assumptions necessary for product-form solution. This assumption is an approximation.

With this product form assumption, the network can be solved using standard techniques [6, 8, 21] to directly obtain system performance measures such as throughput and response time. Various device utilizations have to be interpreted

carefully. For example, the utilization of the "modified" CPU is the fraction of time a non-serialized customer is present at the CPU. More detailed performance measures, for each device and phase can be obtained by applying the inverse transformations in the reverse order.

6. The Iteration

In general, we do not know the "correct" service time adjustment factors, H_{zi}'s. Therefore, we assume an initial value for them and iteratively solve the network by applying these transformations, and their inverse again and again, until there is no significant change in H_{zi}'s. Note that during iteration, we have to map our approximate product form network into original network. This is an inverse process of steps 2-5 above.

First we separate servers and queues in the server-queue pairs for aggregate serialized phase servers to determine the probability that there is customer in the serialized phase z by $p(m_z=1)=p(n_z\geq 1)$. Then we transform each serialized phase server back into isomorphic network of load-concealed physical servers to determine various state probabilities. For example, for $z = 1, \cdots, Z$,

$$p(n_{zi}=1) = \frac{Y_{zi}}{\sum_{l=1}^{K} Y_{zl}} * p(m_z=1) = \frac{Y_{zi}}{\sum_{l=1}^{K} Y_{zl}} * p(n_z\geq 1)$$

In order to determine, other necessary joint probabilities, we also need the following homogeneity assumption (corresponds to equation 6 of Agrawal and Buzen [1]):

$$p(n_{0i}\geq r \underset{q\in\tau}{\wedge} n_{qi}=1 | n_{zi}=1) = p(n_{0i}\geq r \underset{q\in\tau}{\wedge} n_{qi}=1 | n_z\geq 1)$$

where τ is a set of some serialized processing phases and does not include serialized phase z. It is roughly equivalent of saying that when a serialized phase z is busy (i.e., there is some customer in the serialized phase), the steady state behavior of customers in other phases is independent of this serialized customer's where abouts.

We now have the solution for the network obtained at step 2 after applying the load concealment transformation. This solution can be directly mapped into the solution of the of the primary model.

At this point, we can recompute effective service time adjustment factors (H_{zi}'s). If these new values of H_{zi}'s are almost same as the old values, the iteration has converged and we compute necessary performance measures and stop. Otherwise, the iteration is continued.

8. Conclusions

The metamodeling framework developed in this paper can be an important tool for characterizing and analyzing relationships among alternative mathematical models of a single underlying system. The detailed treatment of the aggregate server method presented here has demonstrated the value of this framework for clarifying the underlying structure of a complex solution procedure and for communicating the crucial aspects of such a structure to others.

Even more significant is the potential use of the metamodeling framework for developing new solution procedures. By enabling analysts to divide the solution process into a series of relatively small steps, and by providing a mechanical technique for concatenating the steps together to form a complete solution, metamodeling provides the analyst with a new method of approaching problems that should be of genuine value to future researchers.

Acknowledgement

We are greatful to Peter Denning for his comments on the original version of this paper.

References

1. Agrawal, S.C. and Buzen, J.P., "The Aggregate Server Method for Analyzing Serialization Delays in Computer Systems," *Proc. ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, Seattle, Aug. 30-Sept. 1, 1982. (Also to appear in ACM TOCS, May 1983.)*.
2. Agrawal, S.C. and Buzen, J.P., "Role of Structural Information in Operational Analysis," Tech. Report, Dept. of Computer Sciences, Purdue University, W. Lafayette, IN 47907(*To Appear*).
3. Bard, Y., "A Model of Shared DASD and Multipathing," *Communications of the ACM* 23(10) pp. 564-583 (1980).
4. Baskett, F., Chandy, K. Mani, Muntz, R. R., and Palacios, F. G., "Open, Closed, and Mixed Networks of Queues with Different Classes of Customers," *Journal of the A.C.M.* 22(2) pp. 248-260 (1975).
5. Brandwajn, A.E., "A Model of a Time-Sharing Virtual Memory System Solved Using Equivalence and Decomposition Methods," *Acta Informatica* 4(1) pp. 11-47 (1974).
6. Bruell, S.C. and Balbo, G., *Computational Algorithms for Single and Multiple Class Closed Queueing Network Models*, Series on Programming and Operating Systems, Elsevier/North-Holland Publishing Co., New York(1980).
7. Buzen, J.P., *Queueing Network Models of Multiprogramming*, Ph.D. Thesis, Harvard University, Cambridge, Mass.(1971).
8. Buzen, J.P., "Computational Algorithms for Closed Queueing Networks with Exponential Servers," *Comm. ACM* 16(9) pp. 527-531 (1973).
9. Buzen, J.P., "Fundamental Operational Laws of Computer System Performance," *Acta Informatica* 7(2) pp. 167-182 (1976).
10. Chandy, K.M., Herzog, U., and Woo, L., "Parametric Analysis of Queueing Networks," *IBM Journal of Research and Development* 19(1) pp. 36-42 (Jan. 1975).
11. Coffman, E.G., Jr., "Analysis of a Drum Input/Output Queue Under Scheduled Operation in a Paged Computer System," *J. ACM* 16(1) pp. 73-90 (Jan. 1969).
12. Courtois, P.J., "Decomposability, Instabilities, and Saturation in Multiprogramming Systems," *Comm. ACM* 18(7) pp. 371-376 (July 1975).
13. Denning, P.J. and Buzen, J.P., "The Operational Analysis of Queueing Network Models," *Computing Surveys* 10(3) pp. 225-261 (Sept. 1978).
14. Frank, Howard, "Analysis and Optimization of Disk Storage Devices for Time-Sharing," *J. ACM* 16(4) pp. 602-626 (Oct. 1969).
15. Jacobson, P.A. and Lazowska, E.D., "Analyzing Queueing Networks with Simultaneous Resource Possession," *Comm. ACM* 25(2) pp. 142-151 (Feb. 1982).
16. Kleinrock, L., "Time-Shared Systems: A Theoretical Treatment," *J. ACM* 14(2) pp. 242-261 (1967).
17. Kleinrock, L., *Queueing Systems, Volume 1: Theory*, John Wiley, New York(1975).
18. Kleinrock, L., *Queueing Systems, Volume 2: Computer Applications*, John Wiley, New York(1976).
19. Moore, C.G., III, "Network Models of Large-Scale Time Sharing Systems," Technical Report No. 71-1 (Ph.D. Thesis), Department of Industrial Engineering, University of Michigan, Ann Arbor, Michigan(April 1971).

20. Potier, D. and Leblanc, Ph., "Analysis of Locking Policies in Data Base Management Systems," *Comm. ACM* 23(10) pp. 584-572 (1980).

21. Reiser, M. and Lavenberg, S.S., "Mean Value Analysis of Closed Multichain Queueing Networks," Report RC 7023, IBM T.J. Watson Research Center, Yorktown Heights, N.Y.(March 1978).

22. Sevcik, K.C., "Priority Scheduling Disciplines in Queueing Network Models of Computer Systems," pp. 565-570 in *Proc. IFIP Congress 77*, North-Holland Publishing Co., Amsterdam(1977).