A Comparison of Performance Petri Nets and Queueing Network Models

by

Mary Vernon, John Zahorjan and Edward D. Lazowska

Computer Sciences Technical Report #669

September 1986

A Comparison of Performance Petri Nets and Queueing Network Models

Mary Vernon

Department of Computer Science
University of Wisconsin

John Zahorjan and Edward D. Lazowska

Department of Computer Science
University of Washington

Abstract

Queueing networks (QNs) and performance Petri nets (PPNs) are two approaches to answering performance questions about computer systems. While QNs were originally designed to model resource contention among independent jobs and PPNs were designed to model parallel systems containing synchronization, there is clearly some overlap in the systems that can be modeled using each. In this paper we address this overlap with the goal of indicating whether the two models are fundamentally the same, or whether there are intrinsic differences that make one approach more powerful than the other for particular application domains.

There seem to be two characteristics of a modeling approach that are most important in determining how it may be profitably employed. The first is notation: what models can be expressed, and perhaps more importantly, how convenient is it to specify a particular class of models? The second feature is the evaluation technique: what performance measures can be computed from the model, and what computational effort is required to do so? Our comparison of QNs and PPNs therefore concentrates on these two aspects.

It is conceivable that the shortcomings of either approach in a particular environment might be addressed by adopting features of the other better suited to that environment. We consider this possibility for making improvements to the notations and evaluation efficiencies of the two modeling approaches.

1. Introduction

Queueing networks[1] (QNs) are well established as practical tools in computer system performance analysis and capacity planning [Lazowska et. al. 1984]. The principal reason for their success is the combination of expressive power and solution efficiency that they afford. The notation used to describe the models allows them to be

This material is based upon work supported by the National Science Foundation under Grants DCR-8402680, DCR-8451405, DCR-8302383 and DCR-8352098, and by IBM, DEC, Tektronix, INC., and Weyerhaeuser Co.

Authors' addresses: Mary Vernon, Department of Computer Science, University of Wisconsin, 1210 Dayton Street, Madison, WI 53706. John Zahorjan and Edward D. Lazowska, Department of Computer Science FR-35, University of Washington, Seattle, WA 98195.

[1]We use the term "queueing networks" to denote informally the class of queueing models for which exact or approximate analysis techniques have been developed. These are basically separable queueing networks with specific extensions.

developed very rapidly: only a relatively few parameters are required and these correspond directly to components of the system being modeled. Further, the notation restricts the allowable model structures to those for which efficient (exact or approximate) analytic solution techniques exist. This efficient analysis makes possible interactive tools with subsecond response time, enabling the user to explore a large design space rapidly. The extensive empirical evidence that has been gained demonstrates that the models are sufficiently accurate for the applications in which they are typically employed.

The principal limitation of QNs is that they do not have a general construct for representing synchronization. Specific constructs and associated efficient solution techniques have been developed to represent certain synchronization constraints of importance in the traditional application domain of these models (for instance, memory constraints in centralized systems). However, greater generality is becoming increasingly important with the emergence of parallelism as a central theme in current and future computer systems.

Performance Petri nets (PPNs) are a newer approach to system performance modeling and analysis that are attracting increasing interest [Molloy 1982, Ajmone Marsan et. al. 1984, Holliday and Vernon 1985a, Ajmone Marsan and Chiola 1986a]. PPNs are extensions of *Petri nets* (PNs), which were initially proposed as tools for analyzing the correctness of asynchronous parallel systems. A construct for representing synchronization is fundamental to the notation used to express both sorts of Petri nets. This construct allows for representation of very general synchronization constraints. PPNs differ from PNs in that they include information about residence times and decision probabilities. The timing distributions and/or the structure of the net's state space are generally restricted so that the net defines a discrete state Markov model (in either continuous or discrete time). Thus, the standard analysis techniques for this class of model can be applied to the PPN to compute performance measures of interest. This approach has been applied successfully to the study of several important multiprocessor performance questions, for example [Vernon and Holliday 1986].

The major problem with PPNs is one of efficiency. They naturally include synchronization structures which are not allowed in QNs, and so require relatively more effort to construct, and the available analysis techniques for them are expensive. Another distinction from QNs is that the *exact* solution is usually obtained for PPNs, primarily because of the lack of results to date on efficient approximate techniques. In contrast to QNs, the solution of all but very small PPNs requires considerable computational resources, which severely limits the set and size of applications for which interactive experimentation is possible. Furthermore, even in batch mode the solution rapidly becomes intractable for models of moderate size.

The development of the notations and solution methods for QNs and PPNs have been driven by different applications, and so have emphasized different approaches. At the same time, there is at least some overlap in their expressive powers, so that many systems that one might want to analyze could be cast in terms of either model. An important question arises as to whether PPNs and QNs are fundamentally the same, or whether there are intrinsic differences that make one approach more powerful than the other. This issue has not been previously addressed, and is the subject of this paper.

There are at least two levels at which we can compare the QN and PPN approaches: as notations for describing systems, and as methodologies for system evaluation. Both QNs and PPNs are basically techniques for describing formal models (typically Markovian models), for analyzing the models, and for constructing useful performance measures from the model solutions. Viewed this way, the most important distinction between them is purely notational: once a formal model has been described, the remaining steps of the process are independent of which technique was adopted initially. However, there is a close relationship among the constructs provided by the notation, the overall "modeling approach" taken by the analyst (i.e., the system features that are represented directly vs. approximately in the model), and the computational class of the model that is likely to be generated for some of its useful applications. Thus, we will compare and contrast QNs and PPNs with respect to both their descriptive power and their evaluation efficiency.

In addition to addressing the fundamental questions posed above, we are interested in the following:

- Beyond the formal capability to specify models, in what problem domains is each approach preferable to the other as a descriptive tool?
- To what extent can the notational complexity or the solution complexity of either approach be reduced?

In Section 2 we define more completely QN and PPN models, as considered in this paper. Section 3 then examines the relative expressive power of these models, while Section 4 compares their evaluation efficiency. In Section 5 we consider "borrowing" ideas from one approach to improve the other. Section 6 summarizes these comparisons.

2. Definitions

The terms "queueing networks" and "performance Petri nets" do not have precise definitions. For the purposes of this paper, we will outline informally the main features of each of these models that will be included in the comparison.

2.1. Queueing Networks

Queueing networks consist of a collection of *general constructs* that are the basic building blocks from which models are constructed. These general constructs are *service centers*, which represent servers and queues, and *customers*, which represent the consumers of service. The semantics of these constructs are application independent, and so provide the modeling approach with a degree of flexibility in representing many different kinds of systems. Customers may be divided into several *classes*, such that all customers within a class are statistically identical with respect to routing probabilities and service demands. Each class may be *open* or *closed.*

QNs also contain a set of *specific constructs* that have been developed in response to the need for convenient representation of features often found in applications of importance. Examples include memory constraints [Brandwajn 1982, Lazowska and Zahorjan 1982] and I/O subsystem features [Bard 1982]. These are higher level features than the general constructs, and so carry significantly more semantic information. This simultaneously simplifies the job of modelers requiring these high level features and limits the scope of application of these constructs.

2.2. Performance Petri Nets

PPNs are derived from *classical Petri nets* (PNs), and share with them a set of *general constructs.* Classical Petri nets consist of a set of *places*, a number of *tokens* that reside at the places, and a set of *transitions* that represent events that cause tokens to move from place to place. (We assume the reader is familiar with the structure and firing rules of classical Petri nets. Details can be found in [Peterson 1981].) An example PN is shown in Figure 1. The example includes one fork transition (t1), one join transition (t5), and one place where conflict can occur (p4).

There is no notion of time or branching probabilities in a classical PN. Thus, they are useful for studying the set of *possible* configurations that the net can achieve, but not for deriving performance measures, which requires knowledge of the rate at which configuration changes occur.

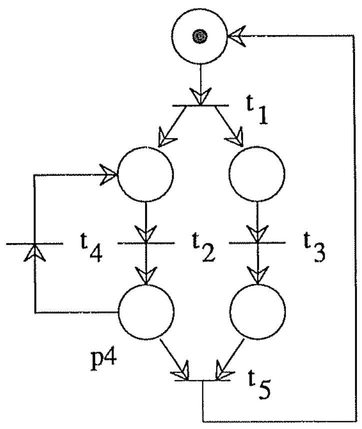


Figure 1: An Example PPN

PPNs are PNs augmented with the notion of time and conflict resolution probabilities. There are two types of PPN models: *stochastic Petri nets* and *timed Petri nets*. The distinction between them is basically that between continuous time and discrete time (Markov) models. There is also a distinction in whether restrictions are imposed on the use of non-zero deterministic firing times in the two types of nets. Because these distinctions are not important here, our discussion of PPNs will allow them to have the properties of either type, as outlined below.

Associated with each transition in a PPN is a probability distribution that governs its "firing time" when it is enabled. For the purpose of evaluation, firing time distributions in PPNs are usually restricted to be Markovian and/or deterministic [Holliday and Vernon 1985a, Ajmone Marsan and Chiola 1986a], but some work has been done with general distributions [Dugan et. al. 1984]. Markov chain techniques can be used to compute steady state performance measures, or probabilities and mean times to absorption, depending on the nature of the model. These techniques are defined only for systems that are *bounded* (i.e., closed).

With the addition of time to these models comes another problem, that of determining which of a number of conflicting transitions should fire. (Conflict occurs in the net when two or more transitions compete for the same input token.) To resolve this, probabilities are associated with the possible outcomes of a conflict in the PPN in one of two ways. The relative probabilities can be state-dependent and explicitly given in the model ("preselection"), or the transition with the smallest sample firing time fires first ("competing rates"). The former specification is more appropriate for conflicting transitions with deterministic or instantaneous firing times, while the latter is often appropriate for conflicts among timed transitions in continuous time models. Explicit probabilities for conflict resolution are indicated next to the appropriate transitions in the net; the absence of explicit probabilities indicates that competing rates are used.

Instantaneous transitions are often useful in PPNs for specifying events whose associated delays do not have a significant impact on system performance. We adopt the graphical notation in [Ajmone Marsan et. al. 1984] to distinguish between these instantaneous transitions and (non-zero valued) timed transitions. Thus, instantaneous transitions are represented by thin bars, as in Figure 1, whereas timed transitions are denoted by thick bars.

3. Expressive Power

A central question in this paper concerns the "expressive power" of PPN and QN notations: what models can be expressed, and how conveniently? Existing work demonstrates that a number of models of interest can be expressed in either notation. Examples include the various models of multiprocessor memory and bus interference that have been expressed in both notations [Goyal and Agerwala 1984, Towsley 1983, Ajmone Marsan et. al. 1984, Holliday and Vernon 1985b]. This observation has motivated the primary question addressed in this section: does either notation dominate the other? This question is addressed in terms of the capabilities for expressing Markovian models, since these are the types of models that are amenable to performance analysis. We defer discussion of the efficiency of solving the expressible models until Section 4.

Of course, it is clear even at first blush that PPNs enjoy an advantage over QNs in representing synchronization. However, there is a natural extension of the QN approach, known as *extended queueing networks* (EQNs), that does incorporate synchronization primitives. In comparing notations, then, we consider EQNs to be the appropriate model of the QN approach[2]. EQNs are defined in Section 3.1 below. Conversely, PPNs can be extended to include a feature known as *colored tokens* [Jensen 1981]. Rules must be given for assigning colors, and for changing colors as tokens move through the net. The number of states in the underlying Markov model grows rapidly with the addition of colored tokens, limiting the practicality of this feature. However, since colored tokens are useful for identifying customer classes and "related tokens", we will consider them to be part of the notation for comparing expressive power with EQNs. The rules for coloring tokens that are "related" are: 1) a unique color is associated with each

[2]While EQNs have a notational advantage over QNs, they suffer significantly in evaluation efficiency. For this reason, it is not entirely fair to substitute EQNs for QNs in this section. However, as we will see, PPNs seem to dominate even EQNs, and for that reason this is not a major concern.

set of tokens that leaves a fork transition, and 2) only tokens of the same color are gathered at the corresponding join transition. A vector of colors must be associated with each customer if nested fork-join operations are allowed. Each fork and join transition must then include a specification of which element in the color vector it operates on.

3.1. Extended Queueing Networks

EQNs, such as the models that can be described in RESQ [Sauer et. al. 1982], allow additional forms of synchronization to be represented. EQNs are QNs that have been augmented with *passive resources*, *fork* and *join nodes*, and *split nodes*. Passive resources are simply tokens that represent the number of units of the resource that are currently available. Special nodes are defined in the extended notation where customers can acquire, release, create or destroy a resource token. If a passive resource token is not available when needed, the customer blocks until another customer makes it available (i.e. creates or releases it). Customer routing in the EQN can depend on the state of any of the resource places or queues in the network. Fork nodes allow a single customer to generate a set of parallel activities, while the join node allows further progress of the original single customer to be synchronized with the completion of all the parallel threads thus generated. Split nodes also generate parallel threads, but these threads do not synchronize at completion. Instead, each thread simply enters a sink node at completion, where it is removed from the model. Figure 2 illustrates the notation we have adopted from RESQ for these constructs.

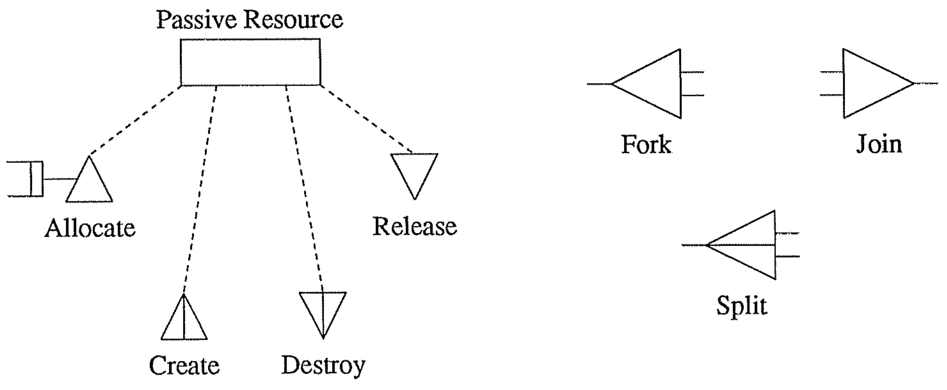


Figure 2: Notation for EQN Primitives

EQNs might also include data variables that can be modified at *set* nodes. Data variables can be defined globally, or associated with an individual customer. (Similar constructs are also possible in Petri Net models.) The inclusion of data variables extends the descriptive power of EQNs to be equivalent to general-purpose simulation languages, and implies that simulation will be used to evaluate the models. Since we are interested in comparing the more formal modeling constructs in the notations that may support analytical solution, we will not consider EQNs or PPNs that include data variables.

3.2. Formal Equivalence

The most obvious question to consider in examining expressive power is what classes of Markov models *can be* described using PPNs or EQNs. As a starting point in addressing this question, we note that *any* finite state Markov chain can be expressed using an extremely degenerate form of either notation. The basic idea is to create a PPN or EQN whose locations have connectivity isomorphic to that of the state space of the Markov model. In other words, a specific place of the PPN or a service center of the EQN would correspond to a single state of the Markov model to be represented. The single token of either model would be used to indicate the current state of the Markov model. The analysis of the PPN or EQN would yield a set of equilibrium "resource utilizations", which would correspond directly to the equilibrium state probabilities of the Markov model. From this, any performance measure of interest could be calculated.

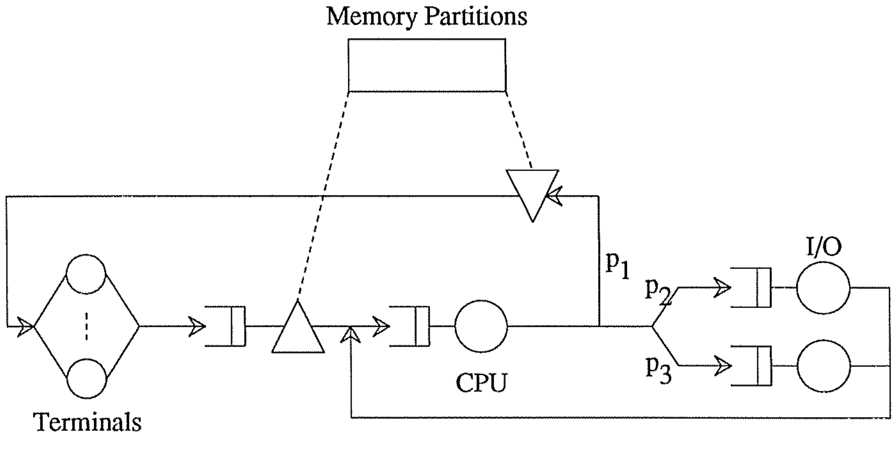
The above proof of equivalence between the notations for expressing continuous time, finite state Markov models is not very satisfying, since the notations have no utility when applied in this way. (Note that the same construction can be applied using simple QNs.) What this construction points out is that even when formally posing the question of the expressive power of the notations we must be concerned with convenience. Thus, a more important question concerns the set of Markov models that can be described "compactly" in each notation. For instance, it seems appropriate to restrict the EQN or PPN to contain no more than $\log N$ components in describing a Markov model with N states. Unfortunately, this appears to be a very difficult problem, and is by no means fully resolved in this paper. Instead, we make some observations in the next section that may be useful in addressing this issue.

3.3. Observations

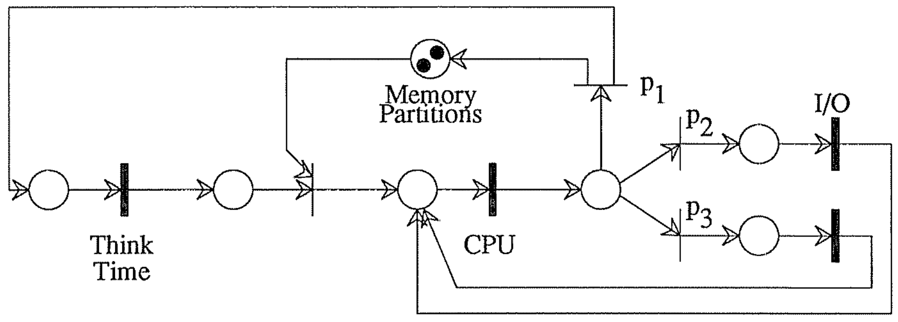
It appears difficult to formally compare PPN and EQN notations with respect to the classes of Markov models that can be specified compactly. However, the following observations provide some evidence that PPNs are more powerful than EQNs in this respect. In particular, observations 1 and 2 imply that any Markov model which can be expressed compactly in the EQN notation can also be expressed compactly in the PPN notation, while observations 3 and 4 indicate that the converse is not true.

Observation #1. Any open, closed, or mixed single class EQN model can be expressed as a PPN of equal complexity.

Figure 3 illustrates this point for a central server model with memory constraints. The construction of an equivalent PPN model from an EQN model proceeds as follows. Each queue and each passive resource place in the EQN is represented by a place in the PPN. Each server, allocate, release, create, and destroy node in the EQN is represented by a transition in the PPN. Routing probabilities in the EQN are represented by decision places (i.e., conflict resolution probabilities) in the equivalent PPN. Fork, join, split, and sink nodes are represented in the obvious way, with colored tokens used in the fork and join operations.



(a) EQN



(b) PPN

Figure 3: Sincle Class EQN and Equivalent PPN

Note that, although the equivalent PPN in Figure 3 has the same (simple) structure that the EQN has, there is less semantic information conveyed in the PPN primitives. For example, the delay-server transition that represents the terminal think time, and the single-server transition that represents the CPU service time, look alike. The server type is denoted by the transition rate (which is marking dependent for the delay-server). As another example, create and release transitions look alike, as do allocate and destroy transitions. The distinction among these respective operations is thus blurred in the PPN. This does not affect the construction or analysis of the PPN model, but does reduce the conceptual information the net diagram conveys to the reader. We will return to this point in Section 5.

Observation #2. Any multi-class EQN model can be represented as a PPN of comparable complexity if we allow scheduling disciplines to be specified for transitions and routing probabilities to be specified per class (i.e., per color).

A multi-class EQN model is defined in the PPN notation by assigning each class a unique color. Passive resources remain "colorless" and can be acquired, released, created, or destroyed by any class of customer. A vector of colors must be associated with each customer if fork and join operations are included in the multiclass model, as was the case for nested fork-join operations. The extensions required to the PPN notation to support multi-class model definitions are straightforward, and the modifications required to evaluate the PPN using Markov chain techniques are minor. Thus we consider PPNs to be capable of expressing these models.

Observation #3. For the class of PPN models that have an equivalent EQN representation, we are not able to define an algorithm for constructing the equivalent EQN model from the PPN.

The difficulty arises because the interpretation associated with a token is not expressed directly in the net. In some very simple cases it is straightforward to give the construction. For example, if the PPN model contains only single-input, single-output transitions, we can replace each transition with a server, each input place with a queue for the server, and each initial token with a customer in the queueing network. Similarly, if a place containing a number of tokens in the initial state of a PPN is only input to a join transition, and only output from a fork transition, we might be able to deduce that the place holds passive resources, the fork transition is a create or release node, and the join transition is an allocate node or an allocate-destroy node pair. Note that, to be safe, the join transition should be replaced by the allocate-destroy node pair and the fork transition by a create node.

When the PPN model includes more complex synchronization constructs, it becomes difficult to identify which of the EQN semantic constructs should be employed in an equivalent EQN model. Without algorithms for translating the basic building blocks, we cannot characterize the classes of PPNs for which equivalent EQNs can and cannot be constructed. We settle instead for some informal examples of PPN models that cannot be represented conveniently and/or accurately in the EQN.

Observation #4. There exist synchronization mechanisms that can be represented in the PPN notation but cannot be represented in the EQN notation.

The synchronization primitives provided in the PPN notation are more flexible than the primitives provided in the EQN notation. A join transition in the PPN can have any number of inputs that may be "related" or not. Each input may have any one of a number of interpretations (e.g., "customer", "passive resource", "arbitrary condition is true"). Furthermore, any input to a join transition may be a decision place (i.e., a place where a token waits for one of several transitions to become enabled, or one of several sets of conditions to be satisfied, to determine the path it will follow).

PPN synchronization primitives allow the local behavior of a particular token to be influenced by essentially arbitrary pieces of the global state information; whereas passive resources can be clumsy to use for this purpose. For example, in a performance model of multiprocessor cache coherency algorithms, whether and how long a local memory request token will wait while the cache responds to a bus request is partially determined by whether another processor's memory request is waiting for a response on the bus. While "well-behaved" systems would not necessarily permit the kinds of unstructured behavior that can be expressed in the PPN, it is sometimes possible to model the important behavior of a well-structured parallel system more abstractly by employing such unstructured mechanisms in the model. These seemingly artificial states introduced in the process are not really artificial but are abstractions of the real state of the system.

A simple example that illustrates the superiority of the synchronization primitives in the PPN for representing behavior of distributed or parallel systems, is the modeling of a "time-out", shown in Figure 4. In the PPN model of Figure 4(a), a token in "wait-event" enables the timer transition. If no token appears in "event" before the timer fires, a time-out event is indicated. Otherwise the "event" token interrupts the timer. Time-outs can only be represented approximately in the EQN. The method shown in Figure 4(b), employs a delay server to represent the timer and a fission node to capture the event [Sauer and MacNair 1983]. The EQN model is approximate because the timer activity must run to completion once it is started.

It should be possible to extend the EQN notation to include synchronization constructs which are similar to the

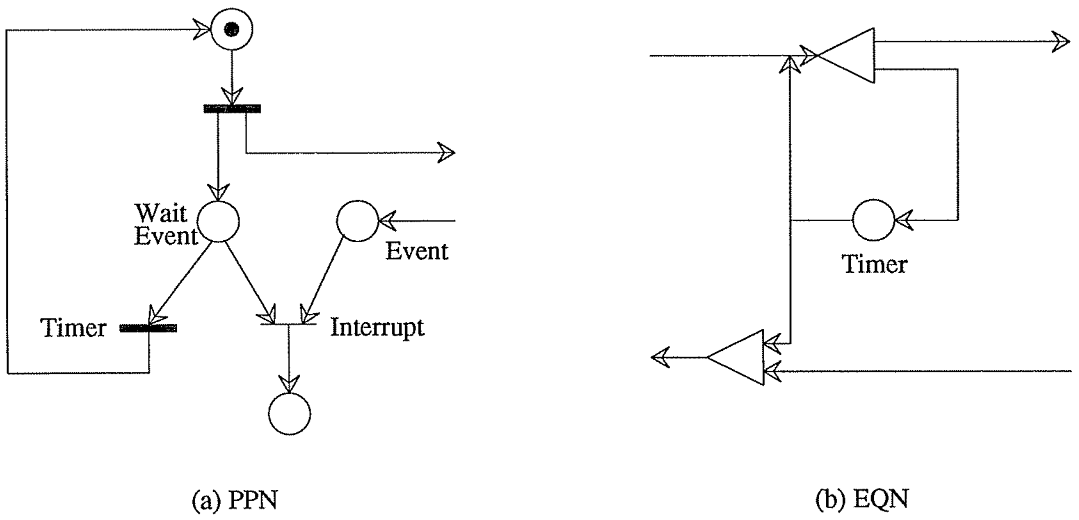


Figure 4: Time-Out Models

general constructs in the PPN. However, given the appropriateness of the PPN notation for the application domains in which these constructs are useful (see Section 5), there do not appear to be compelling reasons for pursuing this idea.

3.4. Summary

The observations above indicate that PPNs have more expressive power than EQNs (and therefore QNs). Moreover, PPNs which are equivalent to QN models have a correspondingly simple structure and are thus as simple to construct. On the other hand, we have not yet addressed the advantages of the restrictive QN notation, which conveys more semantic information to the reader, and guarantees that the model is well-formed and can be solved efficiently. These issues are addressed in Sections 4 and 5.

The added expressive power of PPNs, combined with the principal application of this notation to non-separable systems, has resulted in PPN models that represent synchronization details more directly than QN models. PPNs thus may have the appearance of being more complicated than necessary, particularly to someone who is accustomed to QNs. However, this apparent complexity is the result of the more complicated Markov model that the user is trying to express, and is not primarily an inefficiency of the PPN notation. That is, it is the user's decision to employ that class of Markov model that causes the need for further input parameterization, and so any notation to represent this model would have to provide an equal number of parameters to fully specify the model.

4. Efficiency of Evaluation

The second major criterion important to the applicability of a modeling approach is the evaluation technique used to determine performance measures from the model specification. There are two aspects to this: the performance measures that can be computed and the efficiency of the computation. The first of these properties does not help distinguish between PPNs and QNs. Both are based on equilibrium solutions of Markov models, and so in general both provide only long term performance measures. An application requiring time dependent performance estimates would therefore probably not be well served by either approach, while in the more typical case either approach would probably suffice in this dimension. Thus, we will not consider this aspect further.

The more important distinction between PPNs and QNs concerns the resources required to compute performance measures. PPNs, in general, require resources that are exponential in the size of the model to produce outputs. This means that only models of relatively modest size may be solved at all, and that even for these models execution times are orders of magnitude larger than for QNs. The significance of the exponential nature of the PPN evaluation techniques cannot be overemphasized. It means, for instance, that the cost of evaluating a "typical" multiprocessor performance model can increase by 20-100% each time the number of processors in the model is increased by one. Such rapid growth means that taking one more step up in model size may take the model evaluation cost from the realm of "reasonable" deeply into the realm of "intractable".

5. Technique Comparison

It seems clear that the two primary distinctions between QNs and PPNs are the level at which synchronization can be modeled and the expense of obtaining outputs. Because a primary design consideration in the development of QNs is efficiency (both of construction and evaluation), they have a marked advantage over PPNs in this dimension. In this section we discuss how the features of each modeling technique affect the problem areas to which they are best applied, and comment on how inefficiencies in the PPN approach might be improved by "borrowing" lessons learned from the QN domain.

5.1. Appropriate Application Domains

QNs have established a fairly clear area of primary application, the capacity planning function. They have gained rapid acceptance in this function, and are quite arguably the technique of choice. Although this is not the only application area, it is by far the most important one, and illustrates the aspects of QNs that make them suitable performance modeling tools in many cases. In other words, QNs are best suited to situations applications in which the model inputs contain a measure of uncertainty, such as in projecting workload volumes. In these cases it is usually not sensible to expend a great deal of human or machine resources constructing and evaluating the model, since the limiting factor for accuracy are the input values. While the same models can be expressed in the PPN notation (as shown in Section 3.3), this would be counterproductive since the more restricted QN notation has the added benefit of *guaranteeing* that the model is well-formed (i.e. contains no deadlocks) and can be solved efficiently. Thus, the high level interface and efficient evaluation of QNs are ideal.

On the other hand, PPNs have established the *potential* for being the technique of choice for analyzing parallel system (and parallel algorithm) design alternatives. Again, this is not the only possible application area, but is an important one, and illustrates the strengths of this technique. The notation of PPNs is more general than that of QNs, and so allows greater flexibility in determining the amount of synchronization that is expressed directly in the model. When modeling a parallel system, the analyst is free to choose a level of detail from the high level of QNs to nearly the detail of the implementation. Inputs for the more explicit models often correspond directly to system design constraints and parameters (such as memory read request probabilities or cache hit ratios), and thus in some cases are more readily available than the inputs to QN models that represent the effects of synchronization approximately. Furthermore, the PPN models are probably best suited for design purposes, where the cost of the model can be amortized over a large number of systems or a long system lifetime.

The ability to directly include explicit synchronization features of a parallel system can be an asset not only in representing aspects that clearly have a performance impact, but also in including aspects for which it is not known what the impact might be. Thus, PPNs can provide more reliable results in a new application domain than the QNs. For instance, the manufacturer of a parallel processor might wish to evaluate a number of alternative cache coherency policies. In this case it is necessary to represent the distinctions among the algorithms, and it would be worth considerable computational resources to obtain reliable estimates of the performance of each policy in that architecture. Nevertheless, given the inefficiencies in solving PPN models, it is unclear whether this approach will become the approach of choice for a wide variety of issues in this application domain, or whether the approximate representations possible in QNs will dominate, in spite of potential inaccuracies. If PPNs are to gain the widespread use in the analysis of parallel systems that QNs have gained in the capacity planning arena, it is necessary to find methods for overcoming the inefficiencies of constructing and analyzing these models. In this regard, lessons which have been learned from QNs can provide useful guidance.

5.2. Improving the Notations

In considering the question of improving the notations, it is clear that importing the PPN constructs into QNs would greatly increase their flexibility but at the cost of their primary advantage, solution efficiency. Thus, we do not consider that possibility further.

On the other hand, the QN notation does teach a lesson that can be adopted in PPNs. Speaking broadly, the philosophy of the QN notation is to provide high level primitives that simultaneously supply the user with simple yet expressive building blocks and restricts the models that can be specified to those that can be analyzed efficiently. Petri nets have three modeling primitives that are used to express a wide variety of system behavior. The trend has been to use only these simple syntactic constructs and to leave the conceptual semantics to separate descriptions. For example, transitions which represent "allocate", "allocate and destroy", and "join", are identical in the net diagram. The thin and fat bars that we have employed in this paper to distinguish between instantaneous and timed transitions are an exception, but are not a standard of the notation.

The readability of PPN notation might benefit from the addition of icons that convey more of the semantics of the portion of the model they represent. For instance, tokens and places being used as passive resources could be distinguished from other uses of these same constructs. In general, what we seek is a small set of slightly higher level features that would make it easier for someone examining a PPN to understand what it represents. While this has the flavor of a hierarchical representation, it must be much more restrictive than a general application of that idea, since it is necessary that the new constructs be "standards" if they are to fulfill their roles in increasing the semantic content of the model description. The trick in providing additional semantics to the PPN notation is finding the right level of representation. What is needed are constructs that balance the need to condense routine, complicated structures into more meaningful constructs with the need to access the internal state of those constructs. For instance, priority scheduling might be represented by a distinguished form of transition, using a single icon. However, this would make inaccessible information on the internal state of the priority scheduled transition, whereas a much more complicated representation (e.g., [Balbo et. al. 1985]) would make this information explicitly available.

5.3. Improving the Evaluation Efficiency

It is natural to consider techniques to reduce the cost of evaluating PPNs by considering the approaches that were successfully employed in QNs. In the remainder of this section we mention a number of these, and give a brief indication of how they might be applied to PPNs.

exponential speedups

It is important to note that anything short of an exponential improvement in execution speed can only marginally increase the size of the models that can be solved. For instance, a speedup by a factor of 100 achieved by executing the global Markov analysis on a parallel processor might only increase the maximum model size that can be accommodated by a factor of three or four. For this reason, the most important improvement that could be made would be to find sub-exponential evaluation techniques.

In the QN world this was done by the discovery of product form networks [Jackson 1963, Gordon & Newell 1967, Baskett et al. 1976] and efficient solution techniques for them. In the PPN world, this probably means identifying a restricted class of PPNs that would admit to some more efficient analysis technique. Marsan et al. [1986] have taken a small step in that direction for a specific problem domain, showing that a particular set of PPNs including synchronization in fact admit a product form solution. (However, in general product form PPNs are probably not the right subset to search for, since it seems very likely that the types of synchronization that would be allowed would be much too restrictive.)

An alternative approach to achieving this kind of speedup would be to develop exact or approximate evaluation techniques that dealt solely with the components of the net, and avoided entirely the generation and evaluation of the Markov state space. This requirement seems apparent, because the state space itself has exponential size, and so even its generation rapidly becomes intractable. A first step of this sort has been taken by Ramamoorthy and Ho [1980], who show how to compute minimum cycle times in certain classes of PPNs.

state-space decomposition

Another approach that has been suggested is to speed up the process of computing state probabilities once the state transition rate matrix has been constructed. The theory of nearly-completely decomposable systems [Courtois 1977] might be employed at this level. If this could be done successfully, a decomposition of the state space into n pieces might provide a speedup of about n^2 for this portion of the analysis.

Unfortunately, the lesson taken from QNs is that this is probably not a viable approach. For one thing, exponential work is still required just to create the state space. For another, finding suitable decompositions is potentially more expensive than the straightforward solution technique. Bobbio and Trivedi [1986] have taken a step in this direction which may prove to be fruitful. However, for the above reasons this technique has not become one of practical importance in QNs (although the theoretical foundation that it provides is important to understanding some other techniques of practical consequence).

model level decomposition

Because model evaluation grows exponentially with size, an exponential improvement can be obtained by partitioning the model into pieces, if each can be analyzed independently of the others. The most successful application of this idea in QNs is the use of "flow equivalent service centers" [Lazowska et al. 1984]. This involves three steps: identifying a piece of the model whose behavior might be captured by a simplified representation, obtaining the performance characteristics of that piece, and substituting an appropriately parameterized simplified representation for that piece in the original model.

We have begun work on this approach, with some success [Vernon et al. 1986]. The basic idea is to find pieces of the PPN that have that interact only in very simple ways with the remainder of the model. These pieces are then isolated, and their performance characteristics evaluated using standard techniques (although of course a recursive application of this procedure could be applied to this model). Single timed transitions are then substituted for them in the original model, which is solved by standard methods.

It is important here to note that the cost of this approach is governed by the size of the largest of the models to be solved. Thus, its eventual success is probably limited by the ability to sufficiently partition models of practical interest.

mean value techniques

Perhaps the most important technique employed in QNs is the use of mean values to represent what are in fact dynamic characteristics [Bard 1979]. For instance, in modeling a priority scheduled resource it is common to simply increase the service requirement of the low priority classes to account for the mean resource usage obtained by higher priority classes [Sevcik 1977, Bryant et. al. 1984]; the dynamic pattern of service interruption is completely ignored. It has been observed often that representing complex processes simply by their mean impact has only a negligible effect on accuracy.

This technique might be useful in the PPN context in some situations. For instance, if the PPN can be partitioned into two pieces that interact only through the competition for a passive resource, each piece could be solved in isolation with the effect of the other piece represented by the mean availability of the resource with respect to that piece. This might be done, for instance, by increasing the firing time of the transition that models the successful capture of the shared resource.

ad hoc simplifications

Improvements in evaluation efficiency result any time the size of the model can be reduced. This might be done during the design of the model by careful consideration of the necessary level of detail to be captured. The modeler must be careful not to include aspects of the system that have a negligible impact on performance. In an ideal situation, it might even be possible to achieve a somewhat smooth continuum between efficiency and accuracy by making carefully considered simplifications to the model.

During a series of evaluations of related models, the initial solutions might help in deciding how the models can be simplified. For instance, the solutions of the smaller models might indicate that certain transitions fired only very infrequently. An examination of these transitions might indicate that the portion of the model in which they occur has little effect, and so can be ignored.

6. Conclusions

QNs and PPNs are both "high-level" notations for describing (primarily Markov) models which can be used to answer performance questions about computer systems. We have compared the two notations along the dimensions of *expressive power* and *solution efficiency*, and found each to have a "more powerful" combination of these characteristics than the other for particular applications.

QNs emphasize efficient evaluation for a restricted problem domain, which includes only a few limited forms of synchronization. The QN notation provides convenient primitives for constructing models which can be solved efficiently, and can guarantee that these models are well-formed (i.e. stable and deadlock-free). These models are ideally suited to applications such as capacity planning, in which the effort of constructing more detailed models is not justified, due to the degree of uncertainty in the model inputs. The capability to explore a large design space interactively is an important advantage of this approach.

PPNs have emphasized the flexibility to express important synchronization events directly in the model, at the expense of solution efficiency. This approach can be more appealing when studying new problem domains in which synchronization (actually or potentially) plays a significant role in determining system performance, and where the potential inaccuracy of approximate representation of these effects in the QN model is unacceptable. When modeling a parallel system, the user is free to choose a level of detail from the high level of QNs to a level that corresponds more directly with the implementation of the system under study. The price of this flexibility is the exponential cost of the model solution. Useful results can be obtained, but only for small-to-moderate size systems, and at the expense of considerable computational resources.

QNs are in some sense "optimized" in terms of efficiency of model construction and solution. In contrast, the extent to which PPNs become widely used may depend on whether, or to what extent, the inefficiencies of model construction and solution can be eliminated. Thus, PPNs may benefit from "borrowing" some of the efficient techniques of the QN domain. We have outlined several approaches which might be taken in this direction.

References

[Ajmone Marsan et. al. 1984]
M. Ajmone Marsan, G. Balbo, and B. Conte, A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems. *ACM Trans. on Computer Systems 2,* 2 (May 1984), pp. 93-122.

[Ajmone Marsan and Chiola 1986a]
M. Ajmone Marsan and G. Chiola. On Petri Nets with Deterministic and Exponential Transition Firing Times. *Proc. 7th European Workshop on Application and Theory of Petri Nets* (June 1986), pp. 151-165.

[Ajmone Marsan et. al. 1986b]
M. Ajmone Marsan, G. Balbo, G. Chiola, and S. Donatelli. On the Product Form Solution of a Class of Multiple Bus Multiprocessor System Models. (Private Communication.)

[Balbo et. al. 1985]
G. Balbo, S. C. Bruell, and S. Ghanta. Modeling Priority Schemes. *Proc. 1985 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems* (August 1985), pp. 15-26.

[Bard 1979]
Y. Bard. Some Extensions to Multiclass Queueing Network Analysis. In M. Arato, A. Butrimenko, and E. Gelenbe (eds.), *Performance of Computer Systems*, North-Holland, 1979.

[Bard 1982]
Y. Bard. Modeling I/O Systems with Dynamic Path Selection, and General Transmission Networks. *Proc. ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, 1982, pp. 118-129.

[Baskett et. al. 1975]
F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios. Open, Closed, and Mixed Networks of Queues with Different Classes of Customers. *Journal of the ACM 22,* 2 (April 1975), pp. 248-260.

[Bobbio and Trivedi 1986]
A. Bobbio and K. S. Trivedi. An Aggregation Technique for the Transient Analysis of Stiff Markov Chains. *IEEE Trans. on Computers C-35,* 9 (September 1986), pp. 803-814.

[Brandwajn 1982]
A. Brandwajn. Fast Approximate Solution of Multiprogramming Models. *Proc. ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, 1982, pp. 141-149.

[Bryant et al. 1984]
R.M. Bryant, A.E. Krzesinski, M.S. Lakshmi, and K.M. Chandy. The MVA Priority Approximation. *ACM Transactions on Computer Systems* 2,4 (1984), pp. 335-359.

[Courtois 1977]
P.J. Courtois. *Decomposability*. Academic Press, 1977.

[Dugan et. al. 1984]
J. B. Dugan, K. S. Trivedi, R. M. Geist, and V. F. Nicola, Extended Stochastic Petri Nets: Applications and Analysis. *Proc. 10th Int'l. Symp. on Computer Performance* (December 1984), pp. 507-520.

[Gordon and Newell 1967]
W. J. Gordon and G. F. Newell. Closed Queueing Systems with Exponential Servers. *Operations Research 15* (1967), pp. 254-265.

[Goyal and Agerwala 1984]
A. Goyal and T. Agerwala. Performance Analysis of Future Shared Storage Systems. *IBM J. Res. Develop. 28,* 1 (January 1984), pp. 95-107.

[Holliday and Vernon 1985a]
M. A. Holliday and M. K. Vernon. A Generalized Timed Petri Net Model for Performance Analysis. *Proc. Int'l. Workshop on Timed Petri Nets* (July 1985), 181-190.

[Holliday and Vernon 1985b]
M. A. Holliday and M. K. Vernon. Exact Performance Estimates for Multiprocessor Memory and Bus Interference. Computer Sciences Technical Report #594, Computer Science Dept., University of Wisconsin, Madison, May 1985. (To appear in *IEEE Trans. on Computers.*)

[Jackson 1963]
J.R. Jackson. Jobshop-like Queueing Systems. *Management Science 10* (1963), pp. 131-142.

[Jensen 1981]
K. Jensen. Coloured Petri Nets and the Invariant Method. *Theoretical Computer Science 14* (June 1981), pp. 317-336.

[Lazowska & Zahorjan 1982]
E.D. Lazowska and J. Zahorjan. Multiple Class Memory Constrained Queueing Networks. *Proc. ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, 1982, pp. 130-140.

[Lazowska et al. 1984]
E.D. Lazowska, J. Zahorjan, G.S. Graham, and K.C. Sevcik. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models.* Prentice-Hall, 1984.

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

[Ramamoorthy and Ho 1980]
C. V. Ramamoorthy and G. S. Ho. Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets. *IEEE Trans. on Software Engineering SE-6,* 5 (September 1980), pp. 440-449.

[Sauer et. al. 1982]
C. H. Sauer, E. A. MacNair, and J. F. Kurose. The Research Queueing Package: Past, Present, and Future. *Proc. 1982 National Computer Conference,* AFIPS (1982).

[Sauer and MacNair 1983]
C. H. Sauer and E. A. MacNair. *Simulation of Computer Communication Systems.* Prentice-Hall, 1983.

[Sevcik 1977]
K.C. Sevcik. Priority Scheduling Disciplines in QUeueing Network Models of Computer Systems. *Proc. IFIP Congress '77* (1977), pp. 565-570.

[Towsley 1983]
D. Towsley. An Approximate Analysis of Multiprocessor Systems. *Proc. 1983 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems* (August 1983), pp. 207-213.

[Vernon and Holliday 1986]
M. K. Vernon and M. A. Holliday. Performance Analysis of Multiprocessor Cache Consistency Protocols Using Generalized Timed Petri Nets. *Proc. of Performance '86 and ACM SIGMETRICS 1986 Joint Conf. on Computer Performance Modeling Measurement and Evaluation* (May 1986), pp. 9-17.

[Vernon et. al. 1986]
M. K. Vernon, J. Zahorjan, and E. D. Lazowska. The Use of Flow-Equivalent Servers for Efficient Solution of Performance Petri Net Models. (In preparation.)