A Comparison of Performance Evaluation Process Algebra and Generalized Stochastic Petri Nets

S. Donatelli and M. Ribaudo

Dipartimento di Informatica
Università di Torino
Torino, Italy

J. Hillston

Department of Computer Science
University of Edinburgh
Edinburgh, Scotland

Abstract

Generalized Stochastic Petri Nets (GSPN) and Performance Evaluation Process Algebra (PEPA) can both be used to study qualitative and quantitative behaviour of systems in a single environment.

This paper presents a comparison of the two formalisms in terms of the facilities that they provide to the modeller, considering both the definition and the analysis of the performance model.

Our goal is to provide a better understanding of both formalisms, and to prepare a fertile ground for exchanging ideas and techniques between the two. To illustrate similarities and differences, we make the different issues more concrete by means of an example modelling resource contention.

1 Introduction

In this paper we present a comparison of two formalisms which may be used to develop performance models as continuous time Markov chains (CTMC). Generalized stochastic Petri nets (GSPN) is a well-established high level modelling paradigm which has been widely applied in performance analysis. In contrast, Performance Evaluation Process Algebra (PEPA) is a recently developed formalism. It is a stochastic extension of classical process algebras such as CCS or CSP. PEPA appears to offer several attractive features which have not previously been available to the performance modeller.

As in the paper of Vernon *et al.* comparing queueing networks and performance Petri nets [20], we aim to compare the paradigms in terms of the facilities that they offer the modeller. We consider model construction as well as analysis techniques to produce quantitative and qualitative results.

Note that our motivation is distinct from that of previous studies in which untimed Petri nets and process algebras are investigated in terms of their different representations of causality and concurrency. Here the underlying Markov process forces an interleaving view of concurrency in both paradigms.

Unlike Vernon *et al.* we do not yet aim to determine which applications GSPN and PEPA are particularly suited to. Rather, our intention is to develop a better understanding of both paradigms, individually and in relation to each other, so that such a study may be possible in the future.

The rest of this paper is organised as follows. In Section 2 PEPA is briefly introduced. An introduction to GSPN may be found in the tutorial [1] or in the recent book [2]. The different styles in which the paradigms express the behaviour of systems are examined in Section 3. In Section 4 these points are illustrated by an example. Techniques for quantitative and qualitative analysis for both formalisms are described in Sections 5 and 6 respectively. Finally, in Section 7, we consider possibilities for strengthening each of the paradigms with characteristics, or lessons learned, from the other one.

2 PEPA

The Performance Evaluation Process Algebra (PEPA) is an algebraic description technique based on a classical process algebra and enhanced with timing information. This extension results in models which may be used to calculate performance measures as well as deduce functional properties of the system. In this section we briefly introduce PEPA; more detailed information can be found in [16].

Process algebras are mathematical theories which model concurrent systems by their algebra and provide apparatus for reasoning about the structure and behaviour of the model. In classical process algebras, e.g. Calculus of Communicating Systems (CCS [18]), time is abstracted away—actions are assumed to be instantaneous and only relative ordering is represented—and choices are generally nondeterministic. If an exponentially distributed random variable is used to specify the duration of each action the process algebra may be used to represent a Markov process. This approach is taken in PEPA and the other recently published stochastic process algebras [5, 3, 15]. It is analogous to the association of a duration with the firing of a timed transition in a GSPN model.

The basic elements of PEPA are *components* and *activities*, corresponding to *states* and *transitions* in the underlying CTMC. Each activity is represented by two pieces of information: the label, or *action type*, which identifies it, and the *activity rate* which is the parameter of the negative exponential distribution determining its duration. Thus each action is represented as a pair (α, r). We assume that

the set of possible action types, $\mathcal{A}$, includes a distinguished type, τ. This type denotes internal, or "unknown" activities and provides an important abstraction mechanism.

When enabled, an activity $a = (\alpha, r)$, will delay for a period determined by a sample drawn from the negative exponential distribution with parameter r. As with timed transitions in GSPN, we can think of this as the activity setting a timer whenever it becomes enabled. Similarly, if several (conflicting and/or concurrent) activities are enabled at the same time they are assumed to execute in parallel subject to a *race condition*. The activity whose delay before completion is the minimum will be the one to succeed. An activity may be *interrupted* or *aborted* if another one completes first but the use of the exponential distribution eliminates the need to record previous execution time in the former case.

As in a classical process algebra, the semantics of each term in PEPA is given via a labelled (multi-) transition system (the multiplicities of arcs are significant). In the transition system a state corresponds to each *derivative* (a syntactic term of the language) and arcs represent the actions which can cause one derivative to evolve into another. For any model this semantic construction leads to a *Derivation Graph* (DG) in which actions and their durations are associated with arcs and the nodes are the possible states of the model. This graph is systematically reduced to a form where it can be treated as the state transition diagram of the underlying CTMC. This is analogous to the Reachability Graph (RG) of a GSPN.

PEPA uses a small set of combinators. These allow terms to be constructed defining the behaviour of components, via the activities they undertake and the interactions between them. The combinators, together with their names and interpretations, are presented informally below. A structured operational semantics may be found in [16], and a GSPN semantics in [19].

Prefix, $(\alpha, r).P$ **:** This is the basic mechanism for constructing the behaviours. The component $(\alpha, r).P$ carries out activity (α, r), which has action type α and a duration which is exponentially distributed with parameter r; it subsequently behaves as P.

Choice, $P + Q$ **:** The component $P + Q$ represents a system which may behave either as P or as Q. The activities of both P and Q are enabled. The first activity to complete distinguishes one of them: the other is discarded. The system will then behave as the derivative resulting from the evolution of the chosen component.

Cooperation, $P \underset{L}{\bowtie} Q$ **:** This combinator is in fact an indexed family of combinators, one for each possible set L of visible action types: L is called the *cooperation set*. P and Q proceed independently and concurrently with any activity whose action types is not contained in L (*individual activity*). However, for any activity whose action type is included in L, the components must *cooperate* to achieve the activity (*shared activity*). Such activities are only enabled in $P \underset{L}{\bowtie} Q$ when they are enabled in both P and Q. The rate of the shared activity is altered to reflect the work carried out by both components to complete the activity.

The *apparent rate* of α in P is the total capacity of P to carry out activities of that type, i.e. the sum of the rates of the α type activities enabled. When two components carry out α in cooperation their total capacity to complete α type activities is limited to the capacity of the slower component, i.e. the apparent rate of the shared activity is the minimum of the apparent rate of α in the participating components. Since each component may enable several α activities it must choose which one will take part in the cooperation. We assume that these choices are made independently according to the relative rates. The rate of the shared activity is thus the product of the conditional probability for each participating activity and the apparent rate of the shared action type.

A component may be *passive* with respect to an activity, denoted $(\alpha, \top)$: the rate of the activity is left unspecified until cooperation. The rate of the shared activity is then dictated by the other component. All passive actions must be synchronised in the final model.

If the cooperation set is empty, the two components proceed independently. We use the compact notation, $P \| Q$, to represent this case.

Hiding, P/L **:** Hiding abstracts some aspect of the behaviour of a component so that it is not visible to an external observer, or to other components (i.e. for cooperation). The component P/L behaves as P except that any activities of types within the set L are *hidden*. They appear as the unknown type τ and can be regarded as an internal delay by the component. The rate of the activity is unaffected.

Constant, A **:** Constants are components whose meaning is given by a defining equation: e.g. $A \stackrel{def}{=} P$, which gives the constant A the behaviour of the component P. This is the mechanism of assigning names to components (behaviours).

In order for the Markov process underlying a PEPA model to be ergodic its DG must be strongly connected. A similar condition applies to the RG of a GSPN. Some necessary conditions for ergodicity, at the syntactic level of a PEPA model, have been defined [16]. The class of PEPA terms which satisfy these syntactic conditions are termed *cyclic components*.

3 Descriptive power

In this section we compare GSPN and PEPA from the point of view of model construction. In particular, we examine the different styles in which the formalisms express the behaviour of systems.

At a notational level the difference seems significant since PEPA is a textual language and GSPN has a graphical notation. However, at the other extreme, if we consider the class of Markov processes which can be expressed, it is clear, as in [20], that *any* Markov process can be expressed as a degenerate form of either paradigm. In GSPN a place is associated with each state and appropriate transitions are inserted between the places to represent the transitions in the Markov process. The place corresponding to the initial state is marked by a single token. Similarly for PEPA: a component is associated with the initial state of

the Markov process, with a derivative for each subsequent state; activities capture the transitions.

Neither of these comparisons reflect the modelling styles of the two paradigms, which is what we wish to compare. For example, in the degenerate GSPN the notions of distributed state and local evolution are lost. Therefore, we conduct a more informal comparison, aiming to provide insight into the relative strengths and weaknesses of the two paradigms, without strictly categorising their modelling power.

Five different aspects of modelling style, which highlight the differences between the formalisms, are considered below. The notion of *state* is central to modelling with Markov processes, yet the way that a state is defined in GSPN and PEPA is quite different. Also the model structure in PEPA is forced to be static, whereas in GSPN the entities visible within the model, as represented by tokens, tend to change over time. Both notations have only a few primitives; consequently they both tend to be verbose. At the level of model construction this problem has been tackled differently by the two paradigms. GSPN offers a greater degree of freedom with respect to the chosen level of modelling abstraction. In contrast in PEPA all aspects of a system's behaviour must be modelled explicitly, but the ability to define the components separately, compositional construction and abstraction mechanisms, help to alleviate this problem.

3.1 State vs action orientation

GSPNs have a very clear notion of state, the distribution of tokens in the places of the net. We can regard a GSPN model as an association of a state (the initial marking) to a graph structure, where the graph structure specifies how the state is modified (state evolution). Note that there are two distinct languages: one to define the structure, and one to define the state.

In general, different initial markings lead to completely different Markov chains, and to models with very different properties. For example, they may or may not exhibit cyclic behaviour, or they may be bounded or not. Nevertheless, if a GSPN model gives rise to an ergodic Markov chain, then any reachable state can be taken as the initial state, and the same Markov chain will, of course, be generated.

There is no notion of state in the syntax of PEPA. However, at the semantic level, in the labelled transition system, a state is associated with each syntactic term. Thus for a PEPA model each derivative of the initial expression representing the model is considered to be a state. These derivatives form the states of the underlying Markov process.

Observe that the model and each derivative are specified in the PEPA language. It is impossible to distinguish syntactically whether a term is a derivative of a model or a model itself. Indeed, in PEPA, the concept of state and model coincide, especially in the case of a cyclic PEPA term (ergodic CTMC): since the DG is strongly connected, all derivatives encapsulate sufficient information, via the semantic rules, to recreate the complete behaviour of the model. This has implications for exploitation of equivalence relations and qualitative analysis of models.

From a modelling point of view, GSPN is focussed on states, while PEPA is focussed on actions. Given an arbitrary marking of a GSPN model it is usually possible, by considering the number of tokens in a given place, to immediately infer the state of the system, e.g. "server is idle." In contrast the information that can be immediately extracted from a PEPA model during its evolution is in terms of the actions which the model (system) could perform e.g. "begin_service" action is enabled. This may implicitly tell us that the server is currently idle. This distinction has consequences for the definition of performance measures (Section 5.1).

$$
\begin{aligned}
Q &\stackrel{def}{=} P_1 + P_2 \\
P_1 &\stackrel{def}{=} (\alpha, \lambda_1).(\beta, \mu).Q \\
P_2 &\stackrel{def}{=} (\alpha, \lambda_2).(\beta, \mu).Q
\end{aligned}
$$

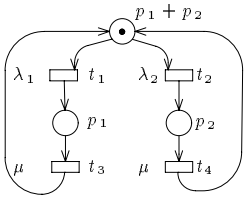


Figure 1: PEPA and GSPN models of a simple system.

To see how the generation of states in the two paradigms differs, consider a simple system in which a job has a choice between two possible evolutions: it may perform action α at rate λ_1 followed by action β at rate μ, or it may choose to perform action α at rate λ_2 followed by action β at rate μ. A representation of this system in the two formalisms is given in Figure 1, by PEPA on the left and GSPN on the right.

The derivative set of the PEPA model has two elements: $\{P_1 + P_2,\ (\beta, \mu).(P_1 + P_2)\}$, while the reachability set of the GSPN model has three states: $\{(1,0,0),\ (0,1,0),\ (0,0,1)\}$. In PEPA, after action (α, λ_1) ((α, λ_2)) takes place, action (β, μ) is executed, without making any distinction of whether β is executed as an action of the first component (P_1) or of the second one (P_2). In the GSPN model instead the two β actions are represented by the two distinct transitions t_3 and t_4 triggered by a condition on two different places. Thus there is a "history" of which component executed the α action remembered by the model. The state transition diagrams of the corresponding Markov processes are shown in Figure 2.

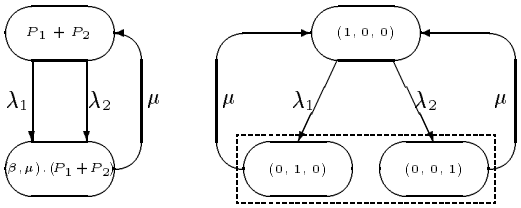


Figure 2: The Markov processes underlying the PEPA and GSPN models respectively.

Note that the states corresponding to the markings $(0,1,0)$ and $(0,0,1)$ of the GSPN form a lumpable subset of states within the Markov process.

This example shows how, in PEPA, terms with a common future are considered to represent the same state, whereas in GSPNs this is not necessarily so. We could have modelled the system with a GSPN where p_1 and p_2 (t_3 and t_4) are fused in a single place (transition), but this does not appear a very "natural" model for the given system, since the two possible evolutions of the job are described separately.

Although the way in which states are derived in GSPN and PEPA differ both can be said to exhibit *distributed* state to some extent. Transitions (activities) need only local knowledge to determine whether they are enabled, and subsequently whether they fire (complete). However, in PEPA only the functional behaviour—whether an activity is enabled—is truly distributed, whereas the temporal behaviour—the expected duration of an activity—is not. The use of apparent rates for assigning rates to shared activities means that such rates may depend upon a global assessment of the state of the system.

3.2 Static vs dynamic model entities

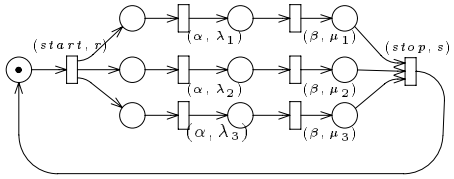
Process algebras distinguish between dynamic and static combinators. Choice is dynamic since after a choice has been made the syntactic form of the component will, in general, be different: e.g. $P + Q \longrightarrow P'$. Cooperation is a static combinator since the syntactic form of the component is the same after evolution regardless of whether a shared or an individual activity was completed, e.g. $P \underset{L}{\bowtie} Q \longrightarrow P' \underset{L}{\bowtie} Q$.

In PEPA, entities within the system are represented as components in the model. In cyclic models the ergodicity condition ensures that the cooperating components of a model are static—they are not created or destroyed as the model evolves. The initial term of a PEPA model shows all the parallel components which are going to exist during the life of the model, and therefore a cyclic PEPA model will have the same number of such terms throughout its evolution. Choices may occur within such components and these represent alternative modes of behaviour but not alternative structures.

In GSPN there is no notion of static components. However, a related concept is that of P*-semiflows* of the net, which are called *conservative* components in Petri net terminology. These are subsets of places such that a weighted sum of tokens in those places is constant for all reachable states. We observe that if a PEPA term is built as the composition of N components, then the equivalent GSPN has at least N P-semiflows where all the places have weight equal to 1.

In a GSPN model, entities within the system are associated with the tokens of the Petri net. As the net evolves the number of tokens may vary, reflecting the dynamic behaviour of the system, according to the firing rule. This models the interactions between the entities in the system. For example, in the case of a GSPN model of a queueing network, we can consider certain tokens as jobs, and others as servers. Tokens representing jobs move from queue to queue through the firing of transitions; when a job is in service this may be represented as a single token although two entities are involved.

For example, consider a simple parallel program containing a `parbegin/parend` section. In the GSPN this can be simply modelled using a Fork & Join structure as shown in Figure 3(a). Representing this system in PEPA produces an alternative view of the system (see Figure 3(b)). The GSPN representation has one process which becomes split into three and then recombined to form a single process again. The static nature of the PEPA models forces a representation with three processes which are initially and finally constrained to act together, only free to act independently during the middle phase of their execution.



(a)

$$
\begin{aligned}
P_{i0} &\stackrel{def}{=} (start, r).P_{i1} \\
P_{i1} &\stackrel{def}{=} (\alpha, \lambda_i).P_{i2} \\
P_{i2} &\stackrel{def}{=} (\beta, \mu_i).P_{i3} \\
P_{i3} &\stackrel{def}{=} (stop, s).P_{i0} \\
System &\stackrel{def}{=} ((P_{10} \underset{\{start,stop\}}{\bowtie} P_{20}) \underset{\{start,stop\}}{\bowtie} P_{30})
\end{aligned}
$$

(b)

Figure 3: GSPN and PEPA models of the Fork & Join structure.

3.3 Modelling abstraction

One of the skills of an experienced modeller is choosing an appropriate level of abstraction at which to construct a model. Although this is largely a question of judgement it is aided by fexibility in the way a system may be presented in a paradigm. GSPN offers more flexibility in this respect than PEPA.

For example, consider a system which is comprised of two identical instances of an entity, which are independent. In PEPA this would be modelled as the component $Q \stackrel{def}{=} P \parallel P$. The component P could have any behaviour but for simplicity we assume that it repeatedly carries out activity (α, r) followed by (β, s): $P \stackrel{def}{=} (\alpha, r).P', \quad P' \stackrel{def}{=} (\beta, s).P$.

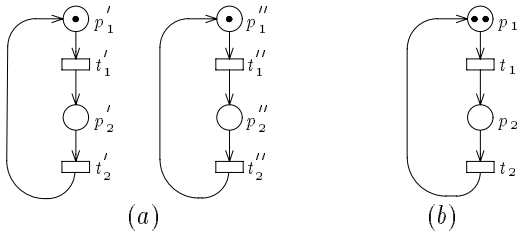


Figure 4: Alternative GSPN representations of two instances of the same entity.

There are two possible GSPN models of this behaviour. In the first, the net representing the behaviour of the entity is constructed, and repeated instances of the component are represented by repeated instances of the same net (Figure 4(a)). Alternatively, the single net structure representing the entity may be marked by two tokens in its initial place to represent the repeated structure (Figure 4(b)), if we assume an infinite server discipline for the transition t_1.

In PEPA repeated instances of the same entity will always be distinguished so all possible interleavings of states must be represented, i.e. we *do* distinguish between $P' \parallel P$ and $P \parallel P'$. There is no way in PEPA of reflecting that the identity of the component which has completed the activity (α, r) is unimportant. In contrast, in the compact GSPN in Figure 4(b) both these cases are represented by the marking $(1, 1)$. This means that GSPN can have a more compact representation than the corresponding PEPA model, and fewer states in the associated Markov chains. It is up to the GSPN modeller to decide whether the more detailed information is needed. Note that again the resulting Markov processes are equivalent up to lumpability.

3.4 Compositionality

Compositionality is a central feature of model construction in PEPA, resulting in models which are easy to understand and readily modified. The expression $Q \stackrel{def}{=} P \parallel P$ shows that the system is comprised of two identical but independent components, without detailed information about the behaviour of P. It follows that a PEPA term may have a lot of embedded behaviour, defined in a separate expression. This leads to a hierarchical approach to model construction. The resulting model has a structure which reflects the structure of the system itself. Moreover this structure may be exploited during analysis. Model components can be developed by different modellers and libraries of re-usable components may be established.

In contrast, with GSPN the model is constructed as a *flat* representation of the system, all modelling primitives conveying the same amount of information.

Observe that in both formalisms it is possible to develop models following a *bottom-up* or *top-down* discipline: the major difference is that PEPA has explicit primitives for this, while GSPN does not.

Compositionality is closely related to the autonomy of components. In cyclic PEPA, due to the restrictions placed on choice, compositionality is based upon cooperation. For example, consider the components P_1 and P_2 defined as:

$$P_1 \stackrel{def}{=} (\alpha, r).P_1 \qquad P_2 \stackrel{def}{=} (\beta, s).P_2$$

The component $P_1 \parallel P_2$ is clearly composed of two components, P_1 and P_2. Even if the cooperation set is not empty as here, the basic behaviour of the two components is established before the composition is formed. If we consider a similar choice of components, Q_1 and Q_2, if the resulting component is to be cyclic, the choice must be recurrent, i.e. whichever component is chosen it must have a derivative which allows the choice to be made again. Thus the individual components have the following form:

$$Q_1 \stackrel{def}{=} (\alpha, r).(Q_1 + Q_2) \qquad Q_2 \stackrel{def}{=} (\beta, r).(Q_1 + Q_2)$$

Here it is clear, even at a syntactic level, that the two components do not have autonomy, i.e. the behaviour of Q_1 cannot be described without reference to Q_2.

Petri nets (timed or untimed) are not compositional in nature, but there have been efforts to add composition operators to the basic, untimed, formalism: composition is based on superposition of places and transitions in [11], while more recently it has been defined in the more structured approach of the Box Calculus [4].

For GSPNs it is not possible, in general, to identify parts of a model which behave autonomously. Special net structures have been established which identify components within the model which have some degree of independence [10]. The motivation for such work is usually to find a decomposition of the model which enables an efficient (exact or approximate) solution of the underlying Markov process. The emphasis of this work is on *decomposition*: models are not constructed compositionally. Recent work on Superposed Stochastic Automata (SSA)and Superposed GSPN (SGSPN) [13] does emphasize compositionality, but from the point of view of solution of the associated Markov process: in both cases the model is defined as a set of interacting components (stochastic automata for SSA and regular GSPN for SGSPN), but this interaction is not defined as part of the basic formalism.

3.5 Operator abstraction

We can consider two forms of operator abstraction in performance modelling paradigms: *functional* and *temporal* abstraction. Functional abstraction allows some behaviour of the system to be abstracted away because it is more detailed than necessary for the current model. Temporal abstraction allows some of the timing information within a system to be abstracted away as irrelevant in the current model. Temporal abstraction is provided by GSPN but not PEPA, whereas functional abstraction is only provided by PEPA.

Temporal abstraction is provided in GSPN by immediate transitions when they are used to represent timed actions, the duration of which is negligible compared with that of other actions within the model. It is assumed that these events do not have a significant impact on the performance of the system. Note that this is distinct from the use of immediate transitions to represent logical actions, to which no time can be associated. As there are only timed activities in PEPA no equivalent form of abstraction is available.

Functional abstraction in PEPA is provided by the hiding operator. Activities of type τ are considered to be internal to the component in which they occur. Thus hiding allows an interface to a model or component to be defined. This is particularly powerful when used in conjunction with the cooperation combinator as it may restrict the interactions of a component. Components of a system may be modelled individually in detail, but subsequently in a more abstract form as the interactions between them are developed. There is no equivalent in GSPN or in any GSPN extensions.

4 Example

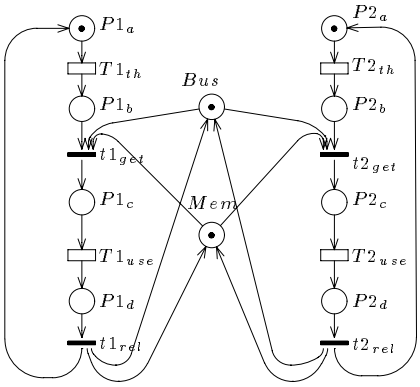
To illustrate the observations of the previous sections we present a simple example of resource contention in a multiprocessor environment. We start with a very simple and abstract model, and then move to more complex ones. Each stage of the example is presented as follows: a short description of the system, its GSPN model, the PEPA model, and a brief comparison.

4.1 The initial system

Consider a multiprocessor system with a shared memory. Processes have to compete for access to the common memory: to gain access and to use the common

memory they need also to acquire the system bus. The bus is released when the access to the common memory is terminated. For simplicity we consider the minimal case of a system composed of two processors, $Proc_1$ and $Proc_2$. All processes have the same functional behaviour, although basic actions progress at different speeds depending on the processor on which they are running. Due to the presence of a single memory and a single bus, the processes may experience undesirable delays. The goal of the model is to study how long the processes wait for access to the common memory. Each process P_i has cyclic behaviour: it performs some local action first, and then accesses the common memory through the bus. Let us consider initially the case of two processes P_1 and P_2 running on $Proc_1$ and $Proc_2$, respectively.

The GSPN model, shown in Figure 5(a), is composed of two subnets representing the cyclic behaviour of P_1 and P_2 and two distinct places representing the bus and the memory respectively. We have chosen not to model processors explicitly. The local "thinking" action in $P_i \ \ (i = 1, 2)$ is modelled by transition Ti_{th}, with rate λ_i, and the use of the memory is modelled by transitions Ti_{use} with rate μ_i. The acquisition and the subsequent release of the memory and the bus are modelled by the immediate transitions ti_{get} and ti_{rel}, respectively. This use of immediate transitions abstracts away from the details of *how* the bus and memory are acquired, and *how long* the acquire and release actions take. The initial marking has one token in each of the places $P1_a$ and $P2_a$, to model the initial state of the two processes, and one token in the Bus and Mem places, to represent the availability of the bus and memory resources.



(a)

$$
\begin{aligned}
P_1 &\stackrel{def}{=} (think, \lambda_1).(get, g).(use, \mu_1).(rel, r).P_1 \\
P_2 &\stackrel{def}{=} (think, \lambda_2).(get, g).(use, \mu_2).(rel, r).P_2 \\
Mem &\stackrel{def}{=} (get, \top).(rel, \top).Mem \\
Bus &\stackrel{def}{=} (get, \top).(rel, \top).Bus \\
Sys &\stackrel{def}{=} (P_1 \parallel P_2) \underset{\{get,rel\}}{\bowtie} Bus \underset{\{get,rel\}}{\bowtie} Mem
\end{aligned}
$$

(b)

Figure 5: GSPN and PEPA models of initial system.

If we model all actions explicitly the PEPA model is as shown in Figure 5(b). Since PEPA does not have immediate actions, we assign a delay to both the get and rel actions. Note that $(get, \top)$ etc. denotes that the rate of the get action in Mem is unspecified since this component is passive with respect to this action.

The number of states of the Markov chains derived from the two models differ slightly, since get and release actions occur in zero time in the GSPN model. For the sake of a fair comparison we shall report the size of tangible and vanishing markings together, indicated as $\|RS\|$. The size of the Derivative Set is denoted by $\|DS\|$. For the models above we have $\|RS\| = 11$ (5 tangible and 6 vanishing) and $\|DS\| = 12$. The additional state occurs because the PEPA model allows the situation where both processes have finished thinking and are trying to acquire the bus and memory. In the GSPN model the higher priority of the immediate transition prohibits this state.

Now consider the system with more than one process in each processor, all processes on the same processor behaving in the same way. In GSPN we can model this new system by simply changing the initial marking of places $P1_a$ and $P2_a$ to be equal to the number of processes in $Proc_1$ and $Proc_2$ respectively.

Similarly the PEPA model is modified by changing the initial term of the model:

$$Sys' \stackrel{def}{=} (\underbrace{P_1 \parallel \cdots \parallel P_1}_{N_1} \parallel \underbrace{P_2 \parallel \cdots \parallel P_2}_{N_2}) \underset{\{get,rel\}}{\bowtie} Bus \underset{\{get,rel\}}{\bowtie} Mem$$

where N_1 and N_2 are the number of processes on $Proc_1$ and $Proc_2$ respectively.

As discussed in 3.3, the state space of the GSPN and PEPA models now differ significantly, for example for 3 processes on $Proc_1$ and 2 on $Proc_2$ we have $\|RS\| = 45$ (18 tangible and 27 vanishing) and $\|DS\| = 192$.

4.2 Identical processes

Let us consider again the initial system, where we have only one process per processor. We now assume that the two processors behave in the same way, even from a timing point of view. Since there is no need to distinguish the processors, we now have slightly simpler GSPN and PEPA models. In particular in the GSPN model the two subnets representing the two processes are folded into a single one (Figure 6(a)), while in the PEPA model we have two replica of P (Figure 6(b)).

However the PEPA model will continue to distinguish between the two processes (unless model simplification techniques are applied); e.g. the derivatives

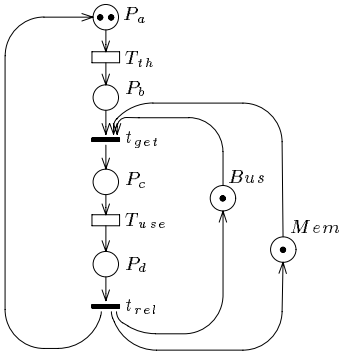
$$(P \parallel (rel, r).P) \underset{\{get,rel\}}{\bowtie} (rel, \top).Bus \underset{\{get,rel\}}{\bowtie} (rel, \top).Mem$$

$$((rel, r).P \parallel P) \underset{\{get,rel\}}{\bowtie} (rel, \top).Bus \underset{\{get,rel\}}{\bowtie} (rel, \top).Mem$$

give rise to distinct states in the underlying Markov process. Thus we can observe that there is a difference in the number of states: $\|RS\| = 6$ (3 tangible and 3 vanishing) while $\|DS\| = 12$.

4.3 Different memory modules

We now consider a more complex example in which there are different memory modules and each process can choose which memory it wants to access. We assume



(a)

$$Sys \stackrel{def}{=} (P \parallel P) \underset{\{get,rel\}}{\bowtie} Bus \underset{\{get,rel\}}{\bowtie} Mem$$

(b)

Figure 6: Compact GSPN model of initial system and simplified PEPA term.

also that the cyclic behaviour of each process is slightly modified: after the access and the release of the common memory, the process performs an update operation of its local state.

In GSPN the choice can be modelled with a conflict of immediate transitions: the weights assigned to these transitions reflect the probabilities of each memory being chosen. The updating operation is modelled by transitions Ti_{up} with rates ν_i (Figure 7(a)).

In the corresponding PEPA model (Figure 7(b)), the choice of which memory to access is made via the race between the two get_{Mi} actions. These two actions, enabled simultaneously although only one of them will succeed, represent acquisition of the chosen memory. Their rates reflect their relative probabilities and the activity rate, i.e. memory $M1$ is chosen with probability p and the rate of a get action remains g.

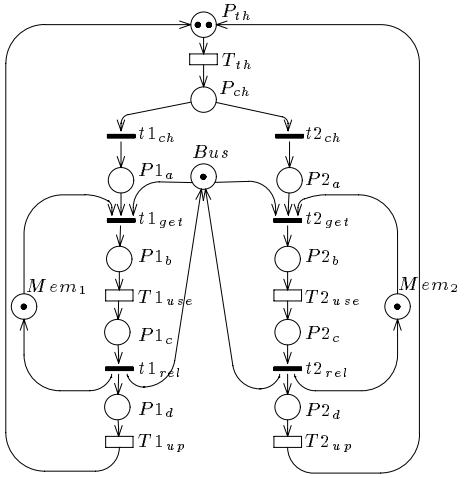
In this case $\|RS\| = 26$ (9 tangible and 17 vanishing) and $\|DS\| = 48$. As before the discrepancy is due to the priority levels present in the GSPN model. If $\nu_1 = \nu_2$ the derivatives $P1$ and $P2$ in the PEPA model become indistinguishable and $\|DS\| = 33$.

4.4 Discussion

We have modelled the same systems using both formalisms and shown that the state spaces underlying the GSPN models are always smaller than those of the corresponding PEPA models.

	$\|RS\|$	$\|DS\|$
Fig. 5	12	12
Fig. 6	7	12
Fig. 7	45	48(33)

Such a comparison is unequal because PEPA does not have immediate actions. If we replace the immediate transitions of the GSPN models in Figures 5(a)-7(a) with timed ones, thus eliminating vanishing states and moving from



(a)

$$\begin{aligned}
P &\stackrel{def}{=} (think,\lambda).((get_{M1}, g \times p).(use,\mu_1).(rel_{M1},r).P1 \\
&\quad + (get_{M2}, g \times (1-p)).(use,\mu_2).(rel_{M2},r).P2) \\
P1 &\stackrel{def}{=} (update,\nu_1).P \qquad P2 \stackrel{def}{=} (update,\nu_2).P \\
M1 &\stackrel{def}{=} (get_{M1},\top).(rel_{M1},\top).M1 \\
M2 &\stackrel{def}{=} (get_{M2},\top).(rel_{M2},\top).M2 \\
Bus &\stackrel{def}{=} (get_{M1},\top).(rel_{M1},\top).Bus \\
&\quad + (get_{M2},\top).(rel_{M2},\top).Bus \\
Sys &\stackrel{def}{=} (P \parallel P) \underset{L}{\bowtie} Bus \underset{L}{\bowtie} (M1 \parallel M2) \\
&\quad \text{where} \quad L = \{get_{M1}, get_{M2}, rel_{M1}, rel_{M2}\}
\end{aligned}$$

(b)

Figure 7: GSPN and PEPA models of the system with two memory modules.

GSPN to SPN, the underlying state spaces are as shown in the table above. Since the RS and the DS only have the same size for the models of the initial system we can conclude that the differences on the sizes of the underlying state spaces are not only due to the presence of immediate transitions.

The model in Figures 6(a) is analogous to that described in Figure 4: there is a single net structure and the repeated instances are modelled by the two tokens in the initial marking. This more compact representation leads to a less detailed state space in which the identity of the component that uses the memory is lost. The model in Figure 7(a) has a smaller state space for the same reason. However, when the rates of the $update$ actions are the same, the PEPA model has 33 derivatives while the RS still contains 45 states. The situation is analogous to that shown in Figure 1: each pair of states that differ only in the marking of places $P1_d$ and $P2_d$ (they can never be marked together) corresponds to a single derivative in the DS.

5 Quantitative analysis

The solution techniques applied to compute quantitative results in GSPN and PEPA are identical, based on the numerical solution of the underlying CTMC. Starting from a GSPN (PEPA) model, the associated Reachability (Derivation) Graph is obtained and then reduced to the corresponding CTMC. This is then numerically solved to compute the steady state probabilities. Tool support is available in both cases.

There has been more work on efficient algorithms for finding and solving this Markov process in the case of GSPNs, and there has been a certain effort towards "less expensive" solution methods, based on product form results [14], computation of bounds [8], approximations [17], and solution based on decomposition [10, 13]. The authors have no doubts that the efficient algorithms for the construction and solution of the associated CTMC can be easily imported into PEPA, and there is also hope that some of the economical solution methods can be transferred. State space explosion is a major problem for models in either paradigm.

To ensure steady state solution the same restriction to ergodic Markov processes must be imposed on both paradigms. Necessary syntactic conditions for a PEPA model to ensure an ergodic Markov process have been identified but these are not sufficient. For some net subclasses, such as *free choice* nets, it is possible to prove ergodicity by solving a set of equations derived from the graph structure of the net [12].

5.1 Performance indices and rewards

Both paradigms base the calculation of performance indices on rewards, although in GSPN the reward structure may be implicit. In PEPA rewards are associated with activities whereas in GSPN rewards may be attached to either transitions or places.

The computation of throughput of activities is straightforward in PEPA, while in GSPN it requires the modeller to identify the set of transitions that represent the given activity, and then to sum the throughputs over this set. For example, to compute the throughput of accesses to the common memory for the system in Figure 5 in PEPA we assign a reward to the get action, while in GSPN we need to identify all the transitions which represent the action of acquiring the bus and the memory ($t1_{get}$ and $t2_{get}$) and calculate throughput as the sum of the throughput of the two transitions.

Conversely, if, for the same system, we want to know the throughput of the thinking activity in process P_1, it is immediate in GSPN even if $\lambda_1 = \lambda_2$ (throughput of transition $T1_{th}$), while in PEPA we would need to distinguish the think activity in component P_1 from the one in component P_2.

Performance indices which involve "state-based" information are very natural to compute in GSPN. For example, for the same system we can study the probability of process P_1 being delayed because of contention on global memory, by computing the probability of place $P1_b$ containing one token, that is to say the probability of a partial state (a state that has been only defined partially). We can also study mutual exclusion in a probabilistic sense by computing the probability that place $P1_c$ is marked *and* place $P2_c$ is marked. If these places can not be marked together in any reachable state, this probability is zero, and therefore the two transitions $T1_{use}$ and $T2_{use}$, which represent access to global memory, are never enabled together.

In PEPA, since rewards are defined at the syntactic level in terms of activities, such state-based information is difficult to define directly. For example to calculate the probability of process P_1 experiencing contention requires rewards to be associated with activities of separate components rather than the model as a whole. This facility is not currently available.

6 Qualitative analysis

GSPN and PEPA have both evolved from formal system description techniques and so models in either paradigm can be regarded as a functional representation of the system, as well as a performance representation. The choice, in both GSPN and PEPA, to use a distribution with infinite support for the delays, allows functional properties of the timed models to be proved with the same techniques used for their untimed counterpart.

In this section we shall review state space analysis, common to both GSPN and PEPA, structural analysis, typical of nets, and verification and transformation based on equivalences, particular to process algebras. We postpone until Section 7 a discussion of the possibilities of applying structural analysis to PEPA and equivalences to GSPN.

6.1 State space analysis

Graph-based analysis may be used to answer many questions about system behaviour when applied to the Reachability or Derivation Graph. For example, the presence of a deadlock can be checked on the graph by looking for a dead state, i.e. a node with no output arc, and the reachability of a given state M' starting from a state M can be checked by looking for a direct path in the graph connecting the corresponding nodes.

State space analysis techniques are very powerful, since they allow the proof of many properties of interest by inspection of the graph which contains all possible evolutions of the model. In general, they are considered to be very expensive because the space and time complexity of the graph construction algorithm can exceed acceptable limits. However, in the case of GSPN and PEPA models, intended for performance evaluation, construction of this graph is essential in any case to generate the underlying CTMC.

Analysis of the RGs of the GSPN models in Figures 5-7 shows that each model is free from deadlock, and that all transitions are live, implying that they can fire infinitely often. Moreover, the analysis shows that the RGs are strongly connected, and therefore the associated CTMCs are ergodic. The same analysis can be applied to the DGs of the PEPA models.

6.2 Structural techniques (GSPN)

The structural analysis techniques of GSPN allow the investigation of properties that may be proved directly on the structure of the net, regardless of its initial marking. Any property proved structurally is valid for every possible GSPN model obtained by imposing an arbitrary initial marking.

Structural analysis is performed by applying linear algebraic techniques to the matrix description of the GSPN model (the *incidence matrix*). The method leads to the computation of the so-called *P-semiflows* and *T-semiflows*. Covering all places by P-semiflows is a sufficient condition for a model to be bounded. For example, all the places of the GSPN models in Figures 5-7 are covered by P-semiflows, ensuring that their associated RGs are finite. P-semiflows can also sometimes be used to prove mutual exclusion properties. For the model in Figure 5 we can prove mutually exclusive access to the memory by observing that there is a P-semiflow $Mem + P1_c + P1_d + P2_c + P2_d$. In the initial marking the sum of tokens in these places is equal to one, so it will be equal to one in any reachable marking; therefore transitions $T1_{use}$ and $T2_{use}$ are never both enabled.

A T-semiflow identifies a set of transitions such that, starting from any marking M, firing any transition sequence belonging to the set, returns to marking M. The existence of T-semiflows covering all the transitions in the net is a necessary, but not sufficient, condition for the *liveness* of the model.

6.3 Equivalences (PEPA)

The standard process algebra notions of equivalence have been extended into the timed setting and equivalence relations play two important rôles in the analysis of PEPA models. Exhibiting equivalence between alternative models, or between a model and another system description establishes when the observable behaviour of the two are the same. This may verify the model with respect to the system's behaviour, or confirm that one model may be substituted for another. If the new model has a smaller DG, the substitution constitutes a *simplification* which preserves the functional integrity of the model. Alternatively, finding equivalence classes within the DG identifies repeated patterns of behaviour within the evolution of the model. Replacing each equivalence class by one representative constitutes an *aggregation*. If the equivalence relation is well-defined the observable behaviour is unchanged.

Equivalence relations suitable for both purposes have been defined for PEPA [16]. The standard notion of equivalence for Markov processes is based on isomorphism between states. In process algebras equivalence, termed *bisimulation*, is based on the notion of indistinguishability under observation. In PEPA bisimulation relations are defined which observe temporal and functional aspects of an agent's behaviour (*strong bisimulation*) or stochastic and functional aspects (*strong equivalence*). In this latter case the activities performed by the agents may differ but their externally observed behaviour is identical. These equivalence relations are congruences—they are preserved by all the combinators of the language—so the relations are complementary to the compositionality. Simplification and aggregation can be carried out component-wise within a model.

Applying strong equivalence to the Multiprocessor example in Figure 5, with 3 processes running on $Proc_1$ and 2 processes running on $Proc_2$, $\|DS\|$ is reduced from 192 to 46. As previously the difference of one state, when compared to the corresponding $\|RS\|$ (45), is due to the priority of immediate transitions.

After a reward structure has been defined over a PEPA model hiding may be used to make only those actions to which a reward is attached visible. In effect an interface to the model is defined. An equivalence relation, called *weak isomorphism*, has been developed in PEPA which abstracts away from the hidden actions of a model but ensures the integrity of the reward structure [16]. This equivalence can be used for model simplification. Different reward structures define different interfaces for the model, giving rise to different model simplifications.

Although there has been considerable work on notions of equivalence for Petri nets—indeed all the equivalences defined for process algebras have been defined also for Petri nets—this work has only recently been incorporated into the timed setting of GSPN. In [6] defines an equivalence for labelled SPN based on bisiumlation. In [9], Chiola *et al.* define a notion of equivalence between GSPN models, and between GSPN and SPN models. However, the motivation of this work was to prove that for a very large class of GSPN, an equivalent SPN can be constructed. It was not intended that this equivalence relation should be used in the manner described above, for model verification, model simplification or aggregation.

7 Cross-fertilisation

In the previous sections we have examined the similarities and differences of GSPN and PEPA: here we outline instead the characteristics (operators, solution techniques, or others) of each formalism that we hope to successfully import into the other one.

Compositionality: The benefits of compositionality are clearly visible in PEPA. We can envisage two possible ways of introducing compositionality into GSPN. The first is augmenting GSPN with operators taken from PEPA. Alternatively, time could be added to a formalism such as the Box Calculus which, in the untimed setting, represents an effort to incorporate compositionality into Petri nets.

Operator abstraction: Temporal abstraction could be incorporated into PEPA via the inclusion of immediate activities, which are assumed to have negligible duration. This would be analogous to the use of immediate transitions in GSPN, and would result in vanishing and tangible derivatives. We think that the introduction of immediate activities in PEPA can greatly benefit from the experience gained with immediate transitions in GSPN in two important ways. Firstly, from their definition, which has finally led, [7], to the choice of defining probabilities of immediate transitions at a structural level (through the concept of Extended Conflict Sets). Secondly, from the reuse of the algorithms for the construction of the CTMC, algorithms that require the elimination of the vanishing markings.

Functional abstraction could be incorporated into GSPNs via the use of a labelling function to associate a name with each transition. Hiding is then a relabelling which results in more transitions having the distinguished label τ. As well as making reasoning about a model simpler, functional abstraction may facilitate model simplification as explained in Section 6.3. There is a notion of labelling in untimed Petri nets and a transition may be labelled τ (non-observable). This has not yet been imported into timed or

stochastic nets although it should be straightforward to do so. A similar notion occurs in the Box Calculus where τ transitions are removed from the communication interface of the box.

Rewards and performance measures: Experience with GSPN has shown that it is important for the modeller to have a rich language for performance indices, and suggests that the reward structure of PEPA should be enriched, to include the definition of performance indices based on (partial) states. This could be achieved if the reward structure were to exploit the compositional structure of PEPA components. For example, in PEPA we cannot currently express the condition that (α, r_1) *and* (β, r_2) are enabled. However, if the activities arise within separate components within a model, P and Q say, where the system is $P \underset{L}{\bowtie} Q$, it should be possible to express this as the conjunction of two separate component rewards. In other words, we attach reward 1 to any derivative in which P enables (α, r_1), 0 to all others, and attach reward 1 to any derivative in which Q enables (β, r_2), and 0 to all others, then reward 1 will be associated only with those derivatives which enable both (α, r_1) and (β, r_2).

Structural techniques: Structural techniques have proved very powerful for the analysis of GSPN models: since there is no structural (graph) component in PEPA, we can not hope to import this technique into PEPA directly. Nevertheless we hope to be able to define syntactic techniques for PEPA, starting from the basic ideas of $P-$ and $T-$semiflows, based on the syntactic form of the model expression. For example, we observe that in the GSPN translation of a PEPA model [19], each PEPA component will be a P-semiflow of weight and token count equal to one (the converse is not the case—not every P-semiflows will be a PEPA component), while if the PEPA model is cyclic, then the transitions of the GSPN are covered by T-semiflows.

Equivalence relations: It is clear from work in the untimed setting that the notions of equivalence based on bisimulation can be naturally applied to Petri nets as well as process algebras. The useful application of these equivalences within PEPA suggest that there may be benefits in extending such notions into the timed setting of GSPNs. However, for fully comparable benefits to be achieved some form of compositionality will also be needed in GSPN. If functional abstraction is incorporated into GSPN there is scope for model simplification which is sensitive to the context of observation, to be developed from an equivalence relation based on PEPA's weak isomorphism.

Kronecker algebra-based solution: As pointed out in the discussion of quantitative analysis there has been far more work on efficient algorithms for finding and solving the CTMC in the case of GSPNs but it is hoped that some of these algorithms may be imported into PEPA directly. More interestingly it is hoped that the SSA and SGSPN ideas for compositional solution based on Kronecker algebra can be adapted for PEPA models. This would involve characterising the PEPA cooperation combinator as a tensor algebra operaton.

To conclude, we have found that the two formalisms have distinctive strengths and weaknesses. One of the strengths of Petri nets is that causality, conflict and concurrency are clearly depicted within a model and this is true for GSPN models as well. From a performance point of view, GSPNs offer the modeller an explicit notion of state, a flexible approach to modelling abstraction and a rich means of expressing rewards and performance indices. Structural analysis on a GSPN model can provide valuable insight into the behaviour of the system.

In contrast, in process algebra based formalisms such as PEPA, causality is not exhibited and there is no clear notion of state. However, an explicit compositional structure is imposed on the model. This structure makes the model easy to understand, may alleviate problems of model construction and can be exploited for both qualitative and quantitative analysis. In addition, the functional abstraction, offered by τ actions and the hiding mechanism of PEPA, enhances the compositional approach.

GSPN is a much more mature paradigm and this is apparent in the number and scope of GSPN results, and the efficiency of the analysis algorithms. However, the authors are confident that many of these techniques may be imported into PEPA. Furthermore, there is scope for future work incorporating the attractive characteristics of the formalisms, such as structural analysis or functional abstraction, from one paradigm into the other.

Acknowledgements

S. Donatelli and M. Ribaudo are supported in part by the ESPRIT-BRA project No. 7269 "QMIPS" and by the Italian MURST "40%" project. J. Hillston is supported by EPSRC Research Fellowship B/93/RF/1729. This collaboration was made possible by a grant from The British Council and CNR.

References

[1] M. Ajmone Marsan, G. Balbo, G. Chiola, G. Conte, S. Donatelli, and G. Franceschinis. An introduction to Generalized Stochastic Petri Nets. *Microelectronics and Reliability*, 31(4):699–725, 1991.

[2] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. John Wiley & Sons, 1994.

[3] M. Bernardo, L. Donatiello, and R. Gorrieri. Modelling and Analyzing Concurrent Systems with MPA. In U. Herzog and M. Rettelbach, editors, *Proc. of 2nd Process Algebra and Performance Modelling Workshop*, 1994.

[4] E. Best, R. Devillers, and J.G. Hall. The Box calculus: a new causal algebra with multi-level communication. In *Advances in Petri Nets*, volume 609 of *LNCS*, 1992.

[5] P. Buchholz. Compositional Analysis of a Markovian Process Algebra. In U. Herzog and M. Rettelbach, editors, *Proc. of 2nd Process Algebra and Performance Modelling Workshop*, 1994.

[6] P. Buchholz. A notion of equivalence for stochastic Petri nets. In *Proc. of Int. Conf. on Application and Theory of Petri Nets*, Torino, 1995.

[7] J. Campos, G. Chiola, and M. Silva. Ergodicity and throughput bounds for Petri nets with unique consistent firing count vector. *IEEE Trans. on Software Engineering*, 17(2):117–125, Feb 1991.

[8] G. Chiola, J. Campos, J.M. Colom, M. Silva, and C. Anglano. Operational analysis of timed Petri nets and applications to the computation of performance bounds. In *Proc. 5th Int. Workshop on Petri Nets and Performance Models*, Toulouse, 1993. IEEE-CS Press.

[9] G. Chiola, S. Donatelli, and G. Franceschinis. GSPN versus SPN: what is the actual role of immediate transitions? In *Proc. 4th Int. Workshop on Petri Nets and Performance Models*, pages 20–31, Melbourne, 1991. IEEE-CS Press.

[10] G. Ciardo and K.S. Trivedi. A decomposition approach for stochastic Petri net models. *Performance Evaluation*, 1992.

[11] F. De Cindio, G. De Michelis, L. Pomello, and C. Simone. Superposed automata nets. In C. Girault and W. Reisig, (eds), *Application and Theory of Petri Nets*, 1982.

[12] F. DeCesare, G. Harhalakis, J.M. Proth, M. Silva, and F.B. Vernadat. *Practice of Petri Nets in Manufacturing*. Chapman & Hall, 1993.

[13] S. Donatelli. Superposed generalized stochastic Petri nets: Definition and efficient solution. In *Proc. of 15th Int. Conf. on Application and Theory of Petri Nets*, Zaragoza, 1994.

[14] S. Donatelli and M. Sereno. On the product form solution for stochastic Petri nets. In *Proc. of Int. Conf. on Application and Theory of Petri Nets*, Sheffield UK, 1992.

[15] N. Götz, U. Herzog, and M. Rettelbach. Multiprocessor and Distributed System Design: The Integration of Functional Specification and Performance Analysis using Stochastic Process Algebras. In *Performance'93*, 1993.

[16] J. Hillston. *A Compositional Approach to Performance Modelling*. PhD thesis, University of Edinburgh, 1994. CST-107-94.

[17] H. Jungnitz, B. Sanchez, and M. Silva. Approximate throughput computation of stocahstic marked graphs. *Journal of Parallel and Distributed Computing*, 15(3):282–295, July 1992.

[18] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[19] M. Ribaudo. Stochastic Petri nets semantics for stochastic process algebras. In *Proc. of 16th Int. Workshop on Petri Nets and Performance Models*, Durham, NC, 1995.

[20] M. Vernon, J. Zahorjan, and E. Lazowska. A comparison of performance Petri nets and queueing network models. In *Proc. of Int. Workshop on Modelling Techniques and Performance Evaluation*, Paris, 1987.