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

S. Donatelli† J. Hillston‡ M. Ribaudo†

Abstract

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

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 a running example.

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, based on Petri nets, 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.

*Submitted to Sigmetrics'95

†S. Donatelli and M. Ribaudo of Dipartimento di Informatica, Università di Torino, corso Svizzera 185, 10149 Torino, Italy 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, and this collaboration was made possible by a travel grant from The British Council and CNR.

As in the paper of Vernon *et al.* comparing queueing networks and performance Petri nets [19], 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. We will consider when equivalent representations of PEPA models can be developed using GSPN, and vice versa, as a means to understanding the relative strengths of the paradigms.

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 the two formalisms are briefly reviewed. The different styles in which the paradigms express the behaviour of systems are examined in Section 3, together with the circumstances in which a translation from one formalism to the other is possible. 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 The Formalisms

In this section we briefly introduce the two formalisms, GSPN and PEPA. More detailed information about GSPN can be found in the tutorial introduction [1] and in a recent book [20], while PEPA is described in the recent thesis [17] and in [14].

2.1 GSPN

GSPN (Generalized Stochastic Petri Nets) is defined as a stochastic extension of the mathematical model of Petri nets [22]. In particular the class of untimed models corresponding to GSPN is that of Petri nets with weighted and inhibitor arcs, which we will call PN in the following. We will make an informal introduction to both PN and GSPN, while a more formal treatment may be found in, for example, [4].

A PN is a quintuple (P, T, I, O, H), where P is the set of places, T is the set of transitions, and I, O, H, are functions that define weighted input, output and inhibitor arcs between places and transitions. PNs incorporate a notion of (distributed) state which is denoted by a function $M : P \rightarrow I\!N$, called the *marking*. The nets have an associated graphical representation, where places are circles, transitions are bars, input and output functions are weighted arrows, and inhibitor functions are circle headed arrows. The marking is represented by inscribing place p with $M(p)$ "tokens."

Transitions describe events that may modify the system state and the *firing rule* defines the dynamic behaviour of PN models. For example, in the net below, transition t is *enabled* if $M(p_1) \geq n$, $M(p_2) \geq m$, *and* $M(p_3) < h$. Once enabled, transition t can fire consuming n tokens from place p_1, m from p_2 and depositing u tokens into p_4 and v into p_5. The enabling and firing rules highlight the notion of *distributed state*: local information (i.e. on input, inhibitor and output places) is sufficient for modifying the state of the system.

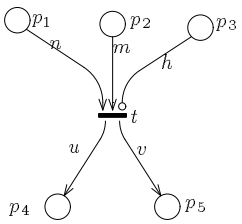


Figure 1: Example of a GSPN model.

Starting from the initial marking, and by firing transitions of the model, it is possible to compute the set of all markings reachable from it (the *Reachability Set* RS) and all the paths that the system may follow to move from state to state (the *Reachability Graph* RG).

PN models can be used for the (qualitative) analysis of *logical* properties of systems. Classical analysis techniques are structural (graph-based) analysis and reachability analysis which investigate, for example, the boundedness of the model or the presence of deadlock.

GSPNs are a timed extension on the basic PN model. Syntactically this extension amounts to adding two functions, π and W. The priority function π assigns a natural number to each transition, and, for the scope of this paper, we can assume that this value is either 0 or 1. *Immediate* transitions, with priority 1, fire in zero time and fire with priority over *timed* transitions (priority 0), which fire only after a certain delay has expired. In GSPN the delay associated with a transition t is a random variable, distributed as a negative exponential, of rate $W(t)$.

The semantics of GSPN is described by a *race model*. When a marking simultaneously enables several (conflicting and/or concurrent) transitions, all activities associated with these transitions are assumed to execute in parallel, so that the next marking

change is due to the transition whose firing delay in the present marking is minimum, i.e., to the transition that wins the race. The firing of the winning transition implies that the activity associated with it in the model is completed. The behaviour of the losing transitions can be specified in different ways. Indeed, it is possible for these transitions to either remember the time during which they have already been enabled (and thus worked), or not. However, the use of exponential distributions for the definition of temporal specifications makes unnecessary the distinction between the distribution of the delay itself, and the distribution of the remaining delay after a change of state, thus avoiding the need for the specification of the behaviour of the transitions that do not fire in a given marking.

Immediate transitions are often useful for specifying events whose associated delays do not have a significant impact on system performance—for instance, the time required to test the condition to enter a *while* loop can be considered negligible with respect to the time required to execute the *body* of the loop; moreover having priority over timed transitions, they can be used to model preemptive conditions. When two or more immediate transitions t_i are in conflict in the same marking (and therefore the firing of one will prevent the other ones from firing) the selection of the one that fires first is done using their weights $W(t_i)$, normalized in such a way as to obtain a discrete probability distribution function.

Due to the presence of immediate transitions, the reachability set of a GSPN model contains two different types of markings that are classified as *tangible* and *vanishing*. A tangible marking is a marking in which only timed transitions are enabled and therefore the system spends some time in that state, while a vanishing marking is a marking in which only immediate transitions are enabled and therefore the time spent in a vanishing marking is equal to zero.

It has been shown [2, 4] that there is a correspondence between GSPN models and continuous time Markov chains (CTMCs). Performance indices such as, for example, transition throughputs and the mean number of tokens in a place, can be associated with a GSPN model. They are computed starting from either the transient or the steady state probabilities of the associated CTMC.

For the Markov chain to be ergodic we need the RG to be strongly connected. Nets whose boundedness and home states' existence can be ensured at a structural level do have an ergodic underlying Markov chain, but in the more general case the condition for ergodicity needs to be checked on the RG.

For the sake of the comparison contained in this paper, it is important to add that the weight associated with a timed transition can be marking dependent: the rate of the transition is then a function of the state of the system. When the function is expressed only in terms of the input, inhibitor and output places of the transition the inherent distributed nature of the state of a GSPN is preserved. However with more general types of dependence the locality of transition firing may be completely destroyed.

2.2 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.

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. Systems are viewed as consisting of *agents* who undertake *actions*. In classical process algebras, such as the Calculus of Communicating Systems (CCS [21]) and the Communicating Sequential Processes (CSP [18]), only the relative ordering of events is modelled and branching is nondeterministic. In PEPA an exponentially distributed random variable, representing *duration*, is associated with every activity in a model. This is analogous to the association of a duration with the firing of a timed transition in a GSPN model.

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 "invisible" 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 makes the distinction unnecessary.

As in a classical process algebra, the semantics of each term in PEPA is given via a labelled transition system[1]. In the transition system a state is associated with each syntactic term of the language and arcs represent the actions which can cause one syntactic term to evolve into another. For any model this semantic construction leads to a *Derivation Graph* 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 of a GSPN.

In PEPA, a system is described as an interaction of *components*. Each component may

[1] Actually a labelled *multi*-transition system since the multiplicities of arcs are significant.

be atomic or may itself be composed of components. Thus the model of the system is constructed from models of components. The grammar of the language defines the ways in which the behaviour of a component may be built up from activities or an interaction of components. The combinators of the language capture the notions of *prefix, choice, cooperation* and *hiding*. The syntax is as follows:

$$P ::= (\alpha, r).P \;\mid\; P + P \;\mid\; P \underset{L}{\bowtie} P \;\mid\; P/L \;\mid\; A$$

In the following paragraphs we give an informal semantics for each of the combinators. Formal semantics in the structured operational style may be found in [17], and a GSPN semantics in [23].

Prefix, $(\boldsymbol{\alpha}, \boldsymbol{r}).\boldsymbol{P}$: Prefix is the basic mechanism by which the behaviours of components are constructed. The component $(\alpha, r).P$ carries out the activity (α, r), which has action type α and a duration which is exponentially distributed with parameter r. The component subsequently behaves as P.

Choice, $\boldsymbol{P + Q}$: 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 these components: the other is discarded. The continuous nature of the probability distributions ensures that the probability of P and Q both completing an activity at the same time is zero. The system will subsequently behave as the derivative of the chosen component.

Cooperation, $\boldsymbol{P \underset{L}{\bowtie} Q}$: 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. This new rate is defined in terms of the *apparent rate* of the action type in each of the components.

The apparent rate of α in P is the total capacity of the component to carry out activities of that type, i.e. the sum of the rates of the α type activities enabled by P. We assume that when two components carry out α in cooperation their total capacity to complete the activity of that type 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 two contributing components. Since each component may enable several α activities it must chooses which such activity takes part in the cooperation. We assume that

these choices are made independently according to the relative rates. Then rate of the shared activity is the product of the conditional probability that each contributing activity is selected by its enabling component and the apparent rate of the action type in the cooperation.

Alternatively a component may be *passive* with respect to an activity, denoted $(\alpha, \top)$. This means that the rate of the activity is left unspecified but upon cooperation the rate of the shared activity is as specified in the other component.

In the special case when the cooperation set is empty, the two components proceed independently. We use the compact notation, $P \parallel Q$, to represent this case.

Hiding, $\boldsymbol{P/L}$: Hiding abstracts some aspect of the behaviour of a component so that it is not visible to an external observer, or to other components. The component P/L behaves as the component P except that any activities of types within the set L are *hidden*, meaning that their type is not visible. Instead 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 by hiding. When a type has been hidden within a component the component cannot participate in a cooperation on that type.

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

Similarly to the condition on the RG of a GSPN to ensure that the underlying Markov process is ergodic, the derivation graph of a PEPA model must be strongly connected if the underlying Markov process is to be ergodic. Some necessary conditions for ergodicity, at the syntactic level of a PEPA model, have been defined [17]. For example, cooperation must be the highest level combinator. The class of PEPA terms which satisfy these syntactic conditions are termed *cyclic components*.

3 The 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 classes of Markov processes which can be expressed, it is clear, as in [19], 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. Only the place corresponding to the initial state is marked, by a single token. Similarly in a PEPA model 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 all notion of distributed state and local evolution is 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 both notations 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. At the end of the section we consider translations between the two paradigms.

3.1 State vs Action Orientation

GSPNs have a very clear notion of state, the distribution of tokens in the places of the net: indeed a GSPN model is specified by the association of an initial state (the initial marking) with a GSPN. 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 assumed as the initial state, and the same Markov chain will, of course, be generated. We can therefore regard a GSPN model as an association of a state to a graph structure, where the graph structure specifies how the state is modified (state evolution). We can observe that we have two distinct languages here: one to define the structure, and one to define the state.

There is no notion of state at the syntactic level in PEPA. However, at the semantic level, in the labelled transition system, a state is associated with each syntactic term.

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

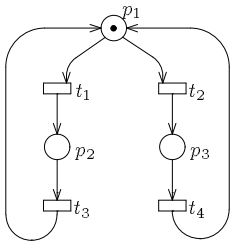


Figure 2: PEPA model and GSPN model of a simple system.

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 both the model and each derivative are specified in the PEPA language, and so 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, which gives rise to an ergodic CTMC: since the Derivation Graph is strongly connected, all derivatives encapsulate sufficient information, via the semantics of the language, to recreate the complete behaviour of the model.

This is substantially different from the implication of ergodicity in GSPN. In nets, given any reachable state of a strongly connected Reachability Graph it is possible, using the information on the structure of the GSPN and the basic firing rule, to build all of the Reachability Graph. In PEPA each state of the derivative set generated from a cyclic model has enough information to be able to build, using only the semantics of PEPA, all the derivative set. This has important implications for the exploitation of equivalence relations and the qualitative analysis of models.

Clearly, from a modelling point of view, GSPN is based on states, while PEPA is based 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 draw some conclusion about 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. a “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).

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 described in Figure 2, by PEPA on the left and by 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, and therefore there is a sort of "history" remembered by the model. The state transition diagrams of the corresponding Markov processes are shown in Figure 3.

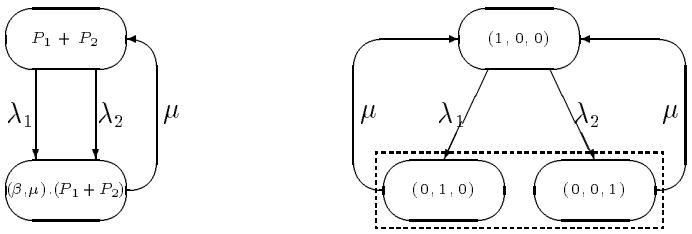


Figure 3: The Markov processes underlying the PEPA and GSPN models respectively.

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. Two Markov processes are defined to be *lumpably equivalent* if there is a lumpable partition of each of them such that there is a one-to-one correspondence between the elements of the partitions which preserves the aggregated transition rates [17]. The Markov processes in Figure 3 are lumpably equivalent if we take the trivial partition of the PEPA process and the partition shown for the GSPN 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_2 and p_3 (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. The automatic translation rules from PEPA to GSPN, described in section 3.6, do produce a GSPN with two distinct places and transitions from the PEPA term, Q, in Figure 2.

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 Modelling 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, differ from before ($P + Q \longrightarrow P'$). 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.

In PEPA entities within the system are represented by components within the model. In cyclic models the syntactic 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 of the GSPN 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. When a transition fires, it removes one or more tokens from its input places and puts one or more tokens in each of its output places. 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. Tokens are seen to *circulate* through the net, although strictly speaking this may not be regarded as the flow of individual tokens.

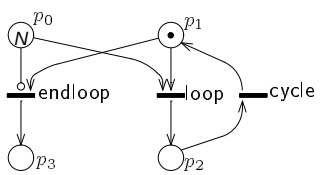


Figure 4: GSPN model of a loop controlled by a variable.

The GSPN model of Fig. 4 models a `while` loop with an associated control variable. The variable is initialized to N, it is decremented by one at each loop, and control leaves the

$$
\begin{aligned}
Process &\stackrel{def}{=} (loop, r).(cycle, s).Process + (endloop, r).Stop \\
Var_i &\stackrel{def}{=} (loop, \top).Var_{i-1} \qquad\qquad 1 \leq i \leq N \\
Var_0 &\stackrel{def}{=} (endloop, \top).Stop \\
Loop &\stackrel{def}{=} Process \underset{\{loop, endloop\}}{\bowtie} Var_N
\end{aligned}
$$

Figure 5: PEPA model of a loop variable.

`while` when the variable reaches zero. In this case we have a "flow of tokens" through places P_1, P_2, and P_3 to represent the program counter of the loop, while we cannot really see a flow of tokens for those tokens modelling the value of the variable.

Since there is no dynamic behaviour explicitly modelled in PEPA there is no notion of flow. Instead the focus is upon the activities of the system and the flow of data, jobs or control associated with those activities is implicit within the model. In the PEPA model (Figure 5) the loop variable is represented as a component which participates in the loop action, each time evolving to a new derivative which represents the current value of the variable.

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 may be aided by a paradigm which offers flexibility in the way that a system may be represented. 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 behave independently. 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 the activity (α, r) immediately followed by (β, s): $P \stackrel{def}{=} (\alpha, r).P', \quad P' \stackrel{def}{=} (\beta, s).P$.

There are two possible GSPN models of this behaviour. In the first case the net representing the behaviour of the entity is constructed, and the repeated instances of the component are represented by repeated instances of the same net structure. This is shown on the left hand side of Figure 6. Alternatively, as shown on the right hand side of Figure 6, the single net structure representing the entity may be marked by two tokens in its initial place to represent the repeated structure, if we assume an infinite server discipline at the transition t_1. Indeed this last model is the usual one for repeated entities.

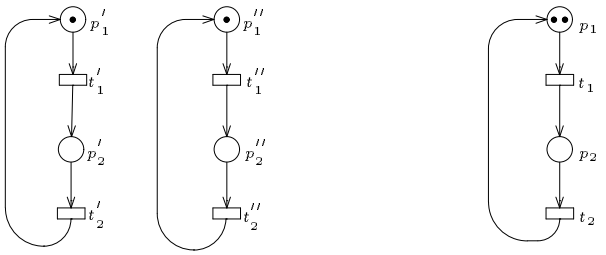


Figure 6: Alternative GSPN representations of two instances of the same entity.

In PEPA even 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 $P \parallel P'$ in the example above. There is no way of reflecting in the PEPA model that the identity of the component which has already completed the activity (α, r) is unimportant. In contrast, in the single structure GSPN in Figure 6 both these cases are represented by the single marking $(p_1 + p_2)$. 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 lumpably equivalent.

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 meaning, or 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 of the model. Models of components can be developed by different modellers and libraries of re-usable components may be developed. 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 general PEPA this is provided by both the choice and the cooperation combinators. However in cyclic PEPA, due to the restrictions placed on choice, compositionality is based solely upon cooperation. For example, consider the components P_1 and P_2 defined:

$P_1 \stackrel{def}{=} (\alpha, r).P_1 \quad P_2 \stackrel{def}{=} (\beta, s).P_2$. 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. On the other hand 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. which ever component is chosen it must have a derivative which allows the choice to be made again. Thus the individual components must have the 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 [8], while more recently it has been defined in the more structured approach of the Box Calculus [13].

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 (system) which have some degree of independence [7]. 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. Note that the emphasis of this work is on *decomposition*: models are not constructed compositionally. Recent work on Superposed Stochastic Automata (SSA) [10] and Superposed GSPN (SGSPN) [11] does emphasize compositionality, but from the point of view of compositional solution of the associated Markov process: although in both cases the model is defined as a set of interacting components (stochastic automata for SSA and regular GSPN for SGSPN), this interaction is never defined as part of the basic formalism.

3.5 Operator Abstraction

It is possible identify 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 from 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 from as being 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. These transitions have a dual rôle, representing both logical actions, to which no time can be associated,

and timed actions, the duration of which is negligible compared with that of other actions within the model. In the latter case it is assumed that the events represented by these transitions do not have a significant impact on the performance of the system. This use of immediate transitions provides temporal abstraction. 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. Thus separate components or aspects of the 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.

3.6 Translation/Relative Expressibility

Any cyclic PEPA model can be represented as a GSPN, possibly with transitions with state dependent rates. Details of the translation can be found in [23] where, following the approach taken by Goltz [15], a compositional net semantics for PEPA has been defined. Starting from the net representations of two components P_1 and P_2 it is possible to derive the net representation of the system $Q \stackrel{def}{=} P_1 \ op \ P_2$ (where op is a syntactic operator of the language) by defining appropriate operations at the net level. For space reasons we cannot describe here the entire procedure and we simply show, in Figure 7, the mapping of each syntactic operator of the language into the corresponding GSPN model. The resulting models are actually SPN, since there are no zero-time activities in PEPA.

In the context of this translation we should emphasise the difference between *transitions* and *actions*: in PEPA different terms can perform actions of the same type and so we can have several transitions representing the same action. On the other hand, by definition, transitions need to be distinguishable, i.e. they must have different names. Therefore, we define a labelling function $l : T \rightarrow \mathcal{A} \times I\!R$ that maps transition names into *(action type, activity rate)* pairs. Note that it is possible that $l(t_i) = l(t_j) = (\alpha, r)$ but $t_i \neq t_j$.

In [23] it has been shown that the translation produces a GSPN that is an SSA, except for the rates of superposed transitions/activities. In SSA transitions rates are computed (statically) while setting up the component models, and therefore depend only on the local state of the component, while apparent rates depend on the number of activities of a given type concurrently enabled, and they need to be computed on the global state.

Each PEPA model can thus be mapped into a GSPN model but the translation from

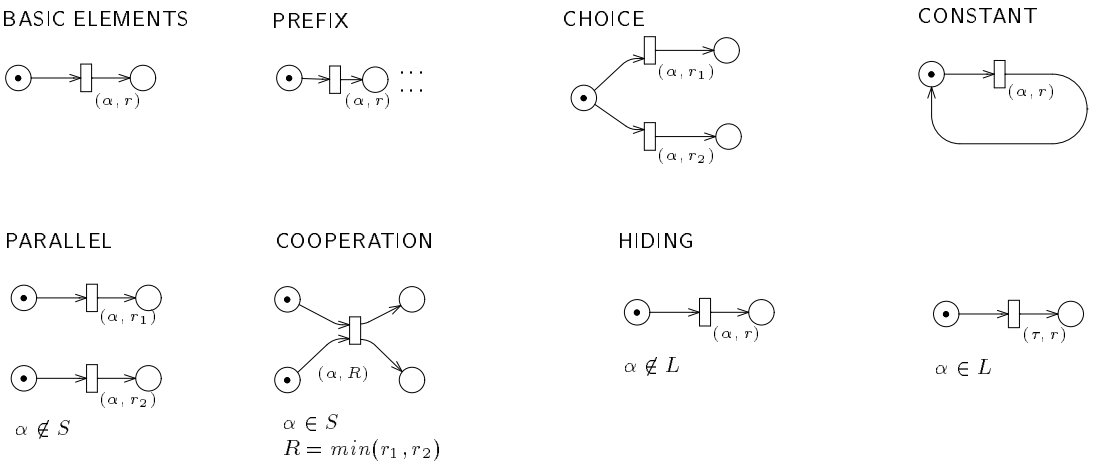


Figure 7: Mapping of PEPA operators into (G)SPN models.

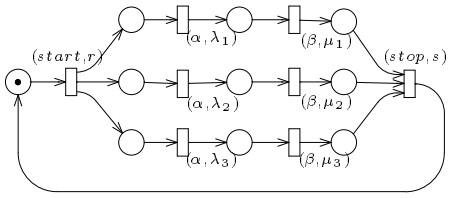


Figure 8: GSPN model of the Fork & Join structure.

a GSPN model to a PEPA model is less straightforward and the resulting model may appear quite different from the GSPN. 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 8.

Representing this system in PEPA produces an alternative view of the system (see Figure 9). 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. It can be argued that this does not truly capture the nature of the program being modelled, although the same underlying Markov process is obtained in both cases.

4 Example

To make the observations of the previous sections more concrete we present a simple example of resource contention in a multiprocessor environment. We will start with a

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

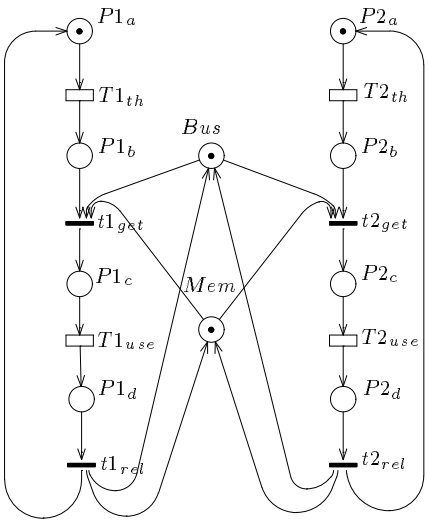
Figure 9: PEPA model of the Fork & Join structure.

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 to be modelled, its GSPN model, followed by the PEPA model, and a brief comparison of the two.

4.1 The initial system

Consider a multiprocessor system with a shared memory. Processes running on this system 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, which we shall call $Proc_1$ 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 10(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 10: GSPN model and PEPA model of the initial system.

If the actions of acquiring and releasing the bus and memory are modelled explicitly the PEPA model is as shown in Figure 10(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

this state is prohibited because of the higher priority of the immediate transition which ensures that one process will successfully acquire the bus and the memory before the other process finishes thinking.

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 $\|DS\| = 192$.

4.1.1

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 in this new system there is no need to distinguish the processors, we 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, as shown in Figure 11, while in the PEPA model we have two replica of a single term P as shown in Figure 11(b).

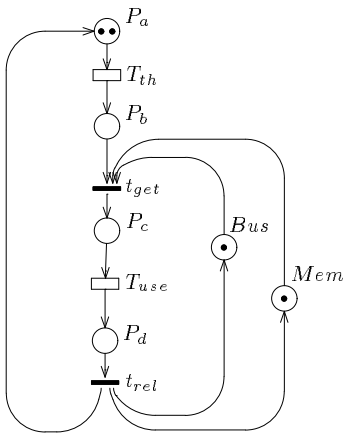
However the PEPA model will continue to distinguish between the two processes (unless model simplification techniques are applied). For example the derivatives

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

and

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

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 $\|DS\| = 12$.



(a)

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

(b)

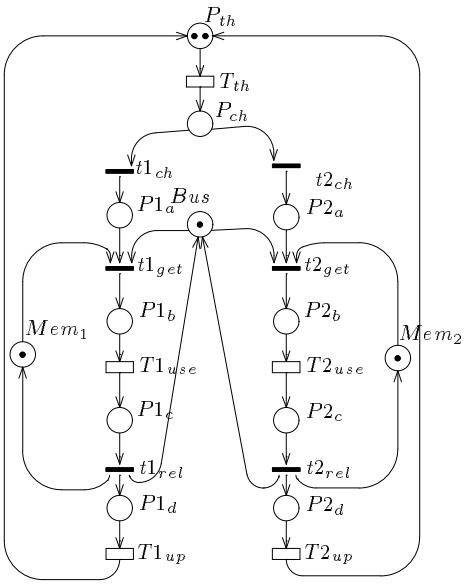
Figure 11: Compact GSPN model of the initial system and simplified PEPA term.

4.1.2

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 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 of the memories can be modelled with a conflict of immediate transitions: the weights assigned to these transitions reflect the probabilities of each of the memory to be chosen. The additional updating operation is modelled by transitions Ti_{up} with rates ν_i. The resulting model is shown in Figure 12(a).

The corresponding PEPA model is shown in Figure 12(b). In PEPA the choice of which memory to access is made via the race between the two get_{Mi} actions. These two actions are enabled simultaneously although only one of them will succeed, representing acquisition of the chosen memory. The rates of these two activities reflect their activity rate and their relative probabilities, i.e. memory $M1$ is chosen with probability p and the rate of a get action is g as previously. In this case $\|RS\| = 26$ (9 tangible and 17 vanishing) and $\|DS\| = 48$. This discrepancy is due to the priority levels present in the GSPN model. If the rates of the update actions in the processors are the same, the derivatives $P1$ and $P2$ in the PEPA model become indistinguishable and $\|DS\| = 33$.



(a)

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

$$\text{where} \quad L = \{get_{M1}, get_{M2}, rel_{M1}, rel_{M2}\}$$

(b)

Figure 12: GSPN and PEPA models of the system with two memory modules.

5 Quantitative Analysis

The solution techniques applied to compute quantitative results in either paradigms are identical, based on the numerical solution of the underlying Markov process. Starting from a GSPN (PEPA) model, the associated reachability (derivation) graph is obtained and then reduced to the correspondent CTMC. The CTMC is then numerically solved to compute the steady state probabilities. Tool support is available in both cases.

At the current time there has been more work on efficient algorithms for finding and solving this process in the case of GSPNs, and there has been a certain effort towards "less expensive" solution methods, based on product form results [12], computation of bounds [5], approximations [16], and solution based on decomposition [7, 11]. 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 hope that some of the economical solution methods can be transferred as well.

The reliance on numerical solution means that both GSPN and PEPA models are prone to *state space explosion*—the large number of states needed to represent the underlying Markov process may make the model intractable.

The same limitations to ergodic Markov processes apply to both paradigms. Necessary syntactic conditions for a PEPA model to ensure an ergodic Markov process have been identified but unfortunately these are not sufficient. For some subclasses of GSPNs, 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 [9].

5.1 Rewards and Performance Measures

Both GSPN and PEPA base the calculation of performance indices on rewards, although in the case of GSPNs 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, to then sum the throughputs over this set. For example, assume we want to compute the throughput of accesses to the common memory for the system presented in section 4.1. In PEPA this only involves assigning a reward to the get action, while in GSPN we need to identify all the transitions of the model of Figure 10 which represent the action of acquiring the bus and the memory, $t1_{get}$ and $t2_{get}$, and the desired throughput is the sum of the throughput of the two activities.

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

Performance indices which involve "state-based" information are very natural to compute in GSPN. For example, for the system in Figure 10 we can study the probability for 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). On the same system 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 in Figure 10 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

For both paradigms graph-based techniques may be used to answer many questions about the system's behaviour. Once the Reachability or Derivation Graph has been built, properties may be checked using classical graph analysis algorithms. For ex-

ample, 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 node corresponding to M to the node corresponding to M'.

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. However, in general, they are 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 to generate the underlying Markov process and therefore is required in any case.

Analysis of the RGs of the GSPN models in Figures 10-12 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 Markov Chains are ergodic. Equivalent results can be obtained by reachability analysis of the transition systems underlying 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 marking on the unmarked net.

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*. P-semiflows are sets of places in which the weighted sum of tokens is constant: covering of the set of places by P-semiflows is a sufficient condition for a GSPN model to be bounded. For example, all the places of the GSPN models in Figures 10-12 are covered by P-semiflows, ensuring that their associated RGs are finite. P-semiflows can sometimes be used to prove mutual exclusion properties. For the GSPN model of Figure 10 we can prove mutually exclusive access to the memory by observing that there is a P-semiflow equal to $Mem + P1_c + P1_d + P2_c + P2_d$. Since in the initial marking the sum of tokens in these places is equal to one, it will also be equal to one in any reachable marking, and therefore transitions $T1_{use}$ and $T2_{use}$ are never enabled together.

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

6.3 Equivalences (PEPA)

Equivalence relations play two important rôles in the analysis of PEPA models, and the formalism has greatly benefited from the extension of the standard process algebra notions of equivalence into the timed setting. Exhibiting equivalence between alternative models, or between a model and another system description (e.g. specification) allows the modeller to establish 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 one model has a smaller derivative set than the other, the substitution constitutes a *simplification* which does not compromise the functional integrity of the model. Alternatively, finding equivalence classes within the derivative set of a model identifies repeated patterns of behaviour within the evolution of the model. Reducing the model to one representation of each such equivalence class constitutes an *aggregation*. Again, the functional and temporal integrity of the model is not compromised if the equivalence relation is well-defined.

Equivalence relations suitable for both purposes have been defined for PEPA [17]. Unlike the standard notion of equivalence for Markov processes, which 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 regard both temporal and functional aspects of an agent's behaviour to be observable (*strong bisimulation*) or both probabilistic and functional aspects (*strong equivalence*). In this latter case the actual transitions performed by the agents may differ but their externally observed behaviour is identical.

Since these equivalence relations are congruences they are preserved by all the combinators of the language. Thus the equivalence relations are complementary to the compositional structure of a PEPA model: model simplification or aggregation may be carried out component-wise within a model.

Applying either of these techniques to the Multiprocessor example in Section 4, when there are 3 processes running on $Proc_1$ and 2 processes running on $Proc_2$, the size of the Derivative Set is reduced from 192 to 46. As previously the difference of one state arises because the priority of immediate transitions over timed transitions in the GSPN prohibits one of the states obtained from the reduced PEPA model.

After a reward structure has been defined over a PEPA model hiding may be used to limit the visible actions of the model to be only those to which a reward is attached. In effect an interface to the model is defined and this limits the contexts in which it can be observed. In [24] Zeigler developed a notion of equivalence between models within limited contexts of observation, called *experimental frames*. In PEPA a similar restricted form of equivalence, called *weak isomorphism*, has been developed which abstracts away from the hidden actions of a model but ensures the integrity of the reward structure [17]. This equivalence may be used for model simplification as outlined

above. Different reward structures can be regarded as different experimental frames 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 not yet been incorporated into the timed setting of GSPN. In [6], Chiola *et al.* define a notion of equivalence between GSPN models, and between GSPN and SPN models. This equivalence implies equivalence of the underlying Markov processes but it is a stronger condition. Since performance indices are often defined at the net level, additional conditions are necessary to ensure that the same performance measures can be derived from the models. However, the motivation of this work was to prove that for a very large class of GSPN, an equivalent SPN can be constructed, thus showing that immediate transitions are usually not necessary. It was not intended that the 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 ([3]) 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, we cannot, in PEPA, 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, 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 but 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.

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. Special issue on Petri nets and related graph models.

[2] M. Ajmone Marsan, G. Balbo, and G. Conte. A class of generalized stochastic Petri nets for the performance a nalysis of multiprocessor systems. *ACM Transactions on Computer Systems*, 2(1), May 1984.

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

[4] G. Chiola, M. Ajmone Marsan, G. Balbo, and G. Conte. Generalized stochastic Petri nets: A definition at the net level and its implications. *IEEE Transactions on Software Engineering*, 19(2):89–107, February 1993.

[5] 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 Intern. Workshop on Petri Nets and Performance Models*, Toulouse, France, October 1993. IEEE-CS Press.

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

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

[8] F. De Cindio, G. De Michelis, L. Pomello, and C. Simone. Superposed automata nets. In C. Girault and W. Reisig, editors, *Application and Theory of Petri Nets*. IFB 52, New York and London, 1982.

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

[10] S. Donatelli. Superposed stochastic automata: a class of stochastic Petri nets with parallel solution and distributed state space. *Performance Evaluation*, 18:21–36, 1993.

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

[12] 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.

[13] R. Devillers E. Best 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.

[14] S. Gilmore and J. Hillston. The PEPA workbench: A tool to support a Process Algebra based approach to performance modelling. In *Proc. Seventh International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, Vienna, 1994.

[15] Ursula Goltz. CCS and Petri nets. Technical Report 467, GMD, July 1990.

[16] B. Sanchez H. Jungnitz and M. Silva. Approximate trhoughput computation of stocahstic marked graphs. *Journal of Parallel and Distributed Computing*, 15(3):282–295, July 1992. Special Issue on Petri Net Modelling of Parallel Computers.

[17] J. Hillston. *A Compositional Approach to Performance Modelling*. PhD thesis, University of Edinburgh, 1994. Ph.D. Thesis.

[18] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[19] J. Zahorjan M. Vernon 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, France, 1987.

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

[21] Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[22] T. Murata. Petri nets: properties, analysis, and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[23] Marina Ribaudo. Understanding Stochastic Process Algebras via their Stochastic Petri Nets Semantics. In *Proc.* 2^{nd} *PAPM Workshop*, Erlangen, 1994.

[24] B.P. Zeigler. *Theory of Modelling and Simulation*. Krieger, 1976.