Specifying Performance Measures for PEPA

Graham Clark, Stephen Gilmore, and Jane Hillston

Laboratory for Foundations of Computer Science,
The University of Edinburgh, Kings Buildings, Edinburgh EH9 3JZ.
Telephone: +44 131 650 1000. Fax: +44 131 667 7209.
Email: {gcla, stg, jeh}@dcs.ed.ac.uk

Abstract. Stochastic process algebras such as PEPA provide ample support for the component-based construction of models. Tools compute the numerical solution of these models; however, the stochastic process algebra methodology lacks support for the specification and calculation of complex performance measures. This paper addresses that problem by presenting a performance specification language which supports high level reasoning about PEPA models, allowing the description of equilibrium (steady-state) measures. The meaning of the specification language can be made formal by examining its foundations in a stochastic modal logic. A case-study is presented to illustrate the approach.

1 Introduction

Performance Evaluation Process Algebra (PEPA) [1] is an expressive formal language for modelling distributed computer and telecommunications systems. PEPA models are constructed by the composition of components which perform individual activities or cooperate on shared ones. To each activity is attached a stochastic estimate of the rate at which it may be performed. Using such a model, a system designer can determine whether a candidate design meets both the behavioural and the temporal requirements demanded of it.

Stochastic process algebras such as PEPA provide ample support for the component-based construction of models. Robust tools such as the PEPA Workbench [2] facilitate the numerical solution of these models when calculating the effective performance of the system under study. However, two important parts of the modelling process are at present insufficiently well supported:

i). the specification and checking of performance properties which are to be satisfied by a model; and
ii). the formulation and calculation of the complex performance measures which are to be derived from the model's numerical solution.

Without additional support for these parts of the modelling process there is a danger that the difficulty of checking correctness and calculating the performance measurements will discourage system designers from undertaking a quantitative analysis. If this were the case, the benefits which are to be gained from a thorough initial investigation of system correctness and responsiveness would be lost.

This paper presents the preliminary results of our attempts to remedy these omissions through the creation of a companion language for PEPA which supports high-level reasoning about PEPA models and provides a suitably-tailored syntax for the description of performance measures over them. This language has foundations in a stochastic logic, the aim being to give a precise definition to specifications in terms of a well-understood theory.

Simple models of a computer system can be constructed without explicit notational support. However, as computer systems become more complex so do their models and the use of a high-level language to aid in their expression becomes necessary. Stochastic process algebras offer attractive features which were not available in previous performance modelling paradigms. The most important of these are *compositionality*, the ability to model a system as the interaction of subsystems, *formality*, giving a precise meaning to all terms in the language, and *abstraction*, the ability to build up complex models from detailed components, disregarding the details when it is appropriate to do so. Queueing networks offer compositionality but not formality; stochastic extensions of Petri nets offer formality but not compositionality; neither offer abstraction mechanisms.

Markovian process algebras are enhanced with information about the duration of activities and, via a race policy, their relative probabilities. Several such languages have appeared in the literature; these include PEPA [1], TIPP [3] and EMPA [4]. Essentially these all propose the same approach to performance modelling: a corresponding continuous time Markov chain (CTMC) is generated via a structured operational semantics; linear algebra can then be used to solve the model in terms of equilibrium behaviour. This behaviour is represented as a probability distribution over all the possible states of the model.

This distribution is seldom the ultimate goal of performance analysis; instead the modeller is interested in performance *measures* which must be derived from this distribution via a *reward structure* defined over the CTMC [5]. A recent case study by first-time users of PEPA [6] reported that a significant proportion of the effort was spent in deriving the performance measures once steady state analysis was complete.

Earlier work by Clark proposed the use of a modal logic to define the reward structure over a PEPA model [7, 8]. While demonstrating feasibility, this work suffered from two major drawbacks. Firstly, the logic used did not include any representation of the timing aspects of PEPA and consequently does not have a clear relationship to the equivalence relations which have been established for the language, such as *Markovian bisimulation* (which is also called *strong equivalence*). Secondly, although the logic formalised an aspect of the PEPA modelling methodology which had previously been carried out in an *ad hoc* manner, it did so in a way which is inaccessible to system designers, the intended users of PEPA. In the current work we aim to address these problems by developing both a stochastic logic which takes full account of the random variables used to represent the duration of activities in PEPA and a model specification language designed to allow the modeller to express, in a high-level way, properties against which the model should be checked.

1.1 Structure of this paper

In the next section we give a succinct summary of the PEPA language and motivate the need for a formal notation for specifying the performance of a PEPA model. Since we provide only a brief summary of PEPA here, the reader should consult [1] for full details. In Section 3 we introduce a formal notation for describing performance measures. This will allow the modeller to make queries about the equilibrium behaviour of a PEPA model. A logical foundation for the specification notation is illustrated in Section 4 where we reveal that the language has a particular relationship to probabilistic modal logic. In Section 5 we illustrate our ideas with a simple, yet realistic, example. Finally, conclusions and future directions for the work are presented at the end of the paper.

2 PEPA

PEPA (Performance Evaluation Process Algebra) extends classical process algebra with the capacity to assign rates to activities, which are described in an abstract model of a system. It is a concise formal language with a small number of grammar rules which define the well-formed terms in the language. An activity of action type α performed at rate r preceding P is denoted by $(\alpha, r).P$. Using the symbol $\top$ instead of a rate denotes *passive* participation in a shared activity. Choices are separated by $+$. Cooperation between P and Q over a set L of action types is $P \underset{L}{\bowtie} Q$ or $P \parallel Q$ if L is empty. Hiding the activities in L and thus denying their availability for cooperation gives the term P/L. The notation for definitional equality is $\stackrel{\mathit{def}}{=}$. The syntax may be formally introduced by means of the following grammar:

$$
\begin{aligned}
S &::= (\alpha, r).S \;\mid\; S + S \;\mid\; C_S \\
P &::= P \underset{L}{\bowtie} P \;\mid\; P/L \;\mid\; C
\end{aligned}
$$

where S denotes a *sequential component* and P denotes a *model component* which executes in parallel. C stands for a constant which denotes either a sequential or a model component, as introduced in a definition. C_S stands for constants which denote sequential components. The effect of this syntactic separation between these types of constants is to constrain legal PEPA components to be cooperations between sequential processes. PEPA is a high-level notation for Markov modelling because it is possible to generate directly from a PEPA model a continuous time Markov chain (CTMC) which faithfully encodes the temporal aspects of the PEPA model.

One reason to fix on a formal notation for a task such as performance modelling is to avoid misunderstanding and misinterpretation of a model. Of course, even when a notation is carefully defined, as PEPA is, there may still be errors of misrepresentation of parts of the system within the model but all of the users of the model can at least agree on the correct interpretation of a given model through recourse to the formal definition of the language. However, when we come to undertake a careful consideration of properties of a model we see that we now have a need for a formal notation for *analysis* of performance models expressed in PEPA. Without this, we would not be able to state performance measures precisely.

It is our intention that each of these specifications should be expressed with reference to a PEPA model. This model is converted to an equivalent encoding as a CTMC for analysis. Different types of analysis of the Markov process may be performed, but for this work, we will require steady-state analysis. This is done through the compilation of the infinitesimal generator matrix of the Markov process and the solution of this by Gaussian elimination or another technique from linear algebra. This leads to a steady state *probability vector*, expressing for each state the long-run probability that the model will be found in that state. For many performance measures, a *reward vector* can be specified, associating particular rewards with particular states. The performance measure can be calculated by the simple scalar product of the probability and reward vectors. For example, if we are considering a lift system, utilisation can be calculated from a steady state analysis, by considering those states in which the lift is utilised to have a reward of 1, and otherwise 0. Timing measures, such as average waiting time, cannot be found directly in this way but can be calculated via an application of Little's Law. However some performance measures cannot be calculated from the steady state information. For example, the percentage of lift requests satisfied in under two minutes requires a *transient analysis*, since we are not interested in long-run probabilities in this case. A preliminary notation for the expression of transient performance measures for PEPA models was given in [9], but is not explored further here.

In earlier work we have considered the problem of specifying reward structures corresponding to PEPA models using a modal logic [7, 8]. However that logic ignored the stochastic elements of the model, namely the random variables used to specify activity durations. In this work we extend work on process logics into this exciting new area, and show links between an established probabilistic logic, and the specification language.

3 A high-level notation for steady state properties

Our objective is to design a high-level notation for expressing steady state properties of a PEPA model. This should provide a straightforward means for the modeller to make quantitative queries about the behaviour of the model, without having to descend into the details of the state space or a reward structure.

Clearly our language must be capable of expressing properties based on those performance measures which can be computed directly by steady state analysis. Typically we wish to know the probability that some condition holds. To express this we use a combination of standard mathematical notation, notation of equivalence relations from PEPA, and a new notation expressing the potential to perform an action of a given type. The use of equivalence relations in this way can focus our queries directly on states, rather than actions. This is unusual within the process algebra literature where notions of state are generally abstracted, and where only potential actions are used to distinguish models. However in Markov processes there is no notion of actions, only states, and all views of the behaviour of a process are phrased in terms of the states it can visit. Stochastic process algebras sit between these two worlds and consequently it is important that our notation has the capabilities to express properties that seem natural within both.

A state based property may mean that we wish to specify the probability that component P is in state P_1—in our notation this is expressed simply as: $\Pr(P = P_1)$. Such a specification is interpreted relative to a model in which P occurs and it succinctly describes the summation of the probabilities of the states of the system where sub-component P is in state P_1. To be pedantic, we should write $\Pr(P \equiv P_1)$ if we intend to require P to be literally P_1 and not just isomorphic to P_1. Similarly, we would write $\Pr(P \cong P_1)$ if we wished to denote the probability that P is in a state which is Markovian bisimilar to P_1. These probabilities may then be used in further calculation such as $r \times \Pr(P = P_1)$ and those results used in comparisons as in $r \times \Pr(P = P_1) > M$. More complex descriptions of states may be expressed via logical operations as in $\Pr(P = P_1 \wedge Q = Q_1)$ or $\Pr(P = P_1 \vee P = P_2)$. For clarity, we may negate relational operators with $\Pr(P \not\equiv Q)$ instead of $\Pr(\neg(P \equiv Q))$.

PEPA allows the modeller to replicate components so that there may be, say, several copies of P in the system description. Thus, we introduce a notion of *situation* (or *location*) of a copy of a component within a PEPA model. It could be the case that the component of interest occurs as a sub-component of another which has only one instance. If so, we use dot notation to identify the sub-component. If not, we number the copies by following a pre-order traversal of the abstract syntax tree of the term.

In order to describe properties in terms of the performance of activities we introduce a term for the probability that a type of activity is enabled. We use the notation $\Pr(\alpha \uparrow)$ for this, whenever the action type of the activity is α. Thus the interpretation of an activity name as a predicate is that the predicate is satisfied whenever the model is in a state S and there is both a state S' and a rate r such that $S \xrightarrow{(\alpha,r)} S'$. A convenient extension to this notation is $\Pr(\alpha \uparrow P)$, meaning that activity α could be performed by component P of the model. However, we shall regard these two forms of *activity probability* as simply convenient abbreviations for a much more complex predicate where the components are constrained (or a given component is constrained) to those states where they may perform an activity of action type α. The cases where the meanings of these two expressions would differ arise whenever the model is not *PEPA live* or not *fully live* [10]. A fully live PEPA model is one such that, for each reachable state, and each syntactic occurrence of an activity within a sequential component, there exists another state which is reachable where this occurrence of the activity can be performed. For the remainder of this paper we restrict consideration to fully live models.

We now have performance measure *expressions* ε, probability *terms* τ, *predicates* ϕ and *situations* σ. These are expressed in the syntax presented in Fig. 1.

$$
\begin{array}{rll}
\varepsilon ::= & \varepsilon + \varepsilon \mid \varepsilon - \varepsilon \mid \varepsilon \times \varepsilon \mid \varepsilon / \varepsilon & \text{(arithmetic expressions)} \\
\mid & \varepsilon \geq \varepsilon \mid \varepsilon > \varepsilon \mid \varepsilon \leq \varepsilon \mid \varepsilon < \varepsilon & \text{(comparison expressions)} \\
\mid & \mathbb{R} \mid \tau & \text{(constants and terms)} \\
\tau ::= & \Pr(\phi) & \text{(probability terms)} \\
\phi ::= & \phi \vee \phi \mid \phi \wedge \phi \mid \neg \phi & \text{(logical operators)} \\
\mid & \sigma \equiv \sigma \mid \sigma = \sigma \mid \sigma \cong \sigma & \text{(local state conditions)} \\
\mid & \alpha\!\uparrow \; \mid \alpha\!\uparrow\!\sigma & \text{(activity predicates)} \\
\sigma ::= & \sigma \underset{L}{\bowtie} \sigma \mid \sigma / L \mid \sigma.C \mid C\#\mathbb{N} \mid C & \text{(situations)}
\end{array}
$$

Fig. 1. Syntax of notation for steady state properties

The characteristics of this notation are that it allows the modeller to inspect internal local states of model components and to consider the steady state probability of attaining significant states of interest. Under the interpretation of activity probabilities as abbreviations for more complex predicates over states we may consider this notation to be entirely based on model states.

4 A logical foundation for the specification language

In this section, we illustrate how the specification language, introduced in Section 3 may be seen to have a formal underpinning, in terms of a probabilistic modal logic. In particular, the expression, and testing for satisfaction of equilibrium properties, can be seen to be closely related to the specification, and model checking of a formula expressed in *probabilistic modal logic* (PML [11]). We give a modified interpretation of such formulae suitable for reasoning about PEPA's continuous time models.

The study of temporal and modal logics in conjunction with models of concurrency is well established. These logics express properties of systems which have a number of states, and in which there is a relation of succession. A modal logic is used to express a finite behaviour. In a temporal logic operators are introduced to allow reasoning over infinite behaviour.

Over the last decade, work on probabilistic verification has accelerated in line with stochastic extensions to models, as used in the performance community. Various modifications to logics allow properties to be expressed which reflect the additional model information. Recent work by Huth and Kwiatkowska [12] gives the temporal logic the *modal mu-calculus* [13] a non-standard semantics, where the meaning of a formula is a function from states to $[0, 1]$. This is intended to express the 'confidence' that a formula is true, for a given state, although the semantics cannot be interpreted directly as probabilities. Their approach results in their model checking procedure giving lower bounds for the probabilities of properties (these may serve the user as a 'guarantee'). Further they note that the work may be modified to deal with *generative* (essentially, autonomous) models such as those described in PEPA. If it was possible to generate true probabilities in this way, it would seem a useful first step in generating performance measures. A differing approach is taken by Hansson and Jonsson [14], whereby probabilistic operators are added to the temporal logic CTL. For example, the formula $[\phi]_{>p}$ is satisfied for a given state if a measure over the set of paths from the state satisfying ϕ is greater than p. Since the formula will either be satisfied by a state, or not, this leads to a classical predicative semantics. A variant of Hansson and Jonsson's model checking procedure is used in the Probabilistic VERUS tool [15]. This is a variant of VERUS, a BDD-based model checker for real-time systems [16]. A VERUS program is a collection of sequential randomised processes; the authors adapt the language to replace non-determinism with discrete probability distributions. With a probabilistic logic similar to that mentioned above, they are able to specify probabilistic properties, and to check whether these properties are satisfied or not. However, as Huth and Kwiatkowska point out, a user's specification of the threshold probability may be inappropriate for the task at hand, where the information required is such a threshold itself. These varying styles of probabilistic verification suggest avenues of exploration for a PEPA reward logic.

An alternative approach to performance evaluation for stochastic process algebras is described by Bernardo [17]. Instead of using a separate logical notation, the author extends the syntax of the stochastic process algebra *EMPA* such that activities include a notion of reward. In order to generate performance measures, a reward structure for the underlying stochastic process is generated; the reward assigned to each state is the sum of the rewards associated with the activities the model enables in that state. By assigning different values (and different interpretations) to the reward field in appropriate activities, one is able to calculate rewards several kinds of performance measure, such as utilisation and throughput. The advantages of the method are that it is relatively simple to use and apply, and is reasonably expressive. In addition, Bernardo has constructed an equivalence relation which respects the additional reward information. An extension of strong Markovian equivalence, for each pair of states in an equivalence class, the total reward accrued by moving into another equivalence class is the same for each state. This will allow the theory of *EMPA* to be extended smoothly to incorporate rewards. However, we can argue that a specification language based on a logical approach too has its advantages. Firstly, it has the potential to be more expressive than an algebra-based technique. By exploiting the operators a modal logic supplies, it is possible to be more discriminating about which states should contribute to the reward measure. In particular, it is possible to select a state based on model behaviour not immediately local to the state. Some examples of such performance measures were presented in [7]. Philosophically, we may also take the point of view that structure for measuring the performance of a model should be separated from the model itself. A disadvantage of a purely logic-based approach is that it may be seen by a user to be esoteric, and requiring of more effort in order to understand and apply. This motivates one of the aims of our current work, to provide a high-level specification language which abstracts from an underlying logic, and provides the user with a framework which is simpler to apply.

We proceed by describing the principle behind using a logic to generate a performance measure. Following that, we highlight why PML may be a suitable logic for our purposes.

4.1 Using logic to specify performance measures

Previous work by Clark [7] proposed an approach to generating measures using traditional Hennessy-Milner logic (HML [18]). The idea was to capture the set of 'interesting' states of the model by partitioning the state space with a formula of the logic—those states that enjoy the property are then assigned a reward, such as a number, or a value based on 'local state' information, such as the rate at which the state may perform a particular activity. All uninteresting states are given a reward of 0. In this way, a reward vector is formally specified, and equilibrium measures such as utilisation and throughput may be calculated. However, the method was not ideal for several reasons. Firstly, it was ad hoc—the logic provided an initial partition only, meaning that a calculational technique was required in addition, in order to assign reward values. Secondly, the logic was qualitative only, in that it disregarded the *rate* at which a PEPA process could perform an activity, and only captured the fact that an activity was possible. We believe these issues can be addressed by using a more appropriate logic, namely Larsen and Skou's PML.

4.2 Probabilistic modal logic

The syntax of PML formulas is given by

$$F ::= \mathtt{tt} \mid \nabla_\alpha \mid \neg F \mid F_1 \wedge F_2 \mid \langle \alpha \rangle_\mu F$$

The models described in [11] are *probabilistic*, in that for any state P and any action α, there is a (discrete) probability distribution over the α-successors of P. Informally, the semantics of a formula ∇_α is the set of states unable to perform an α activity; and the semantics of $\langle \alpha \rangle_\mu F$ is the set of states such that each can make an α-transition with probability *at least* μ to a set of successors each of which satisfies F. We choose to modify slightly the interpretation of these formulae with respect to PEPA models. First we give a simple definition

Definition 1. $P \xrightarrow{(\alpha,\nu)} S$ *if and only if for all* $P' \in S$, $P \xrightarrow{\alpha} P'$, *and* $\sum\{r \mid P \xrightarrow{(\alpha,r)} P', P' \in S\} = \nu$.

Now let P be a model of a PEPA process. Then

$$P \models \texttt{tt}$$
$$P \models \neg F \text{ if } P \not\models F$$
$$P \models F_1 \wedge F_2 \text{ if } P \models F_1 \text{ and } P \models F_2$$
$$P \models \nabla_\alpha \text{ if } P \stackrel{\alpha}{\not\longrightarrow}$$
$$P \models \langle\alpha\rangle_\mu F \text{ if } P \stackrel{(\alpha,\nu)}{\longrightarrow} S \text{ for some } \nu \geq \mu, \text{and for all } P' \in S, P' \models F$$

Therefore, the subscript μ present in formulae of the form $\langle\alpha\rangle_\mu F$ is now interpreted as a rate rather than a probability; if a state P is capable of doing activity α *quickly enough* arriving at a set of states S each of which satisfies F, then P satisfies $\langle\alpha\rangle_\mu F$.

4.3 Relation of PML to the specification language

If we use PML as a vehicle for the semantics of the specification language, it will address one of the criticisms of the original PEPA reward language work—that the logic was badly suited to the models. Now it is possible to distinguish model states that differ only in the rate at which they may perform activities. But how does this logic relate to the specification language? This can be made clearer by first showing the precise nature of the relation of PML to PEPA. In [11], Larsen and Skou show that PML exactly characterises *probabilistic bisimulation*, in the sense that two probabilistic processes are bisimilar if and only if they satisfy exactly the same set of PML formulae. With our modification to the semantics of PML, an analogous result holds for PEPA processes:

Theorem 1 (Modal characterisation of strong equivalence). *Let P be a model of a PEPA process. Then*

$$P \cong Q \textit{ if and only if for all } F, P \models F \textit{ iff } Q \models F$$

That is to say that two PEPA processes are *strongly equivalent* (in particular, their underlying Markov chains are *lumpably equivalent* [1]) if and only if they both satisfy, in our modified setting, the same set of PML formulae.

Instantly, this gives us an understanding of the notation for specifying equilibrium properties. For example, probability terms may take the form $\Pr(P \cong P_1)$. Model checking a logical characterisation of state P_1 via PML would provably capture all and only those states P which are strongly equivalent to P_1, and thus with the steady state vector would immediately lead to a computed value for the term $\Pr(P \cong P_1)$. Another example is provided by the term $\Pr(\alpha\uparrow)$. We can instantly characterise, up to strong equivalence, those states which should be included in the computation of this measure—they are those which satisfy the PML formula $\neg\nabla_\alpha$.

As discussed in Section 3, the performance specification language makes use of the idea of model states, as well as model behaviour *in* a state. This can be smoothly reconciled with the use of a probabilistic logic, and the computation of the reward vector can thus be seen as a two-stage procedure. The method is simple, and standard in the theory of process logics—it is to extend the syntax of PML with a set of *variables* V, and for a given model P with state space S, to extend the semantics with a *valuation* function $\mathcal{V} : V \to 2^S$.

$$F ::= \mathtt{tt} \mid \nabla_\alpha \mid \neg F \mid F_1 \wedge F_2 \mid \langle\alpha\rangle_\mu F \mid X$$

$$P \models X \text{ iff } P \in \mathcal{V}(X)$$

The intuition is that a variable $X \in V$ represents a property which is true in a particular subset of the state space. This allows formulae such as $\neg(\langle\mathtt{transmit}_{120}\rangle FailState)$, where $FailState$ is understood to represent an undesirable portion of the state space—"it is not the case that it is possible to efficiently transmit a network packet and finish in a failure state". Given a model, and an expression given in the specification language, the two stage generation of the reward vector can be understood as:

i). calculating the valuation function according to any 'local state' requirements, by e.g. strong equivalence aggregation [1]; then
ii). computing the satisfaction of the PML formula corresponding to the specification, using a model-checking style technique.

The use of PML suggests ways in which the specification language could be extended, if these features would be useful to users. It will certainly be possible to reason about more complex behaviour than the ability to perform a single activity, and it will be possible to reason directly about the rate at which activities are performed. For example, a user may wish to verify that the percentage of time that a component in his model is transmitting network packets efficiently is higher than 40%. Our notation could be extended to allow $\Pr((\mathtt{transmit}, 120)\uparrow) \geq 0.4$. The states to be included in the computation of the measure would be those captured by the formula $\langle\mathtt{transmit}\rangle_{120}\mathtt{tt}$.

Due to the predicative semantics of PML, we note that it is straightforward to specify utilisation and reliability measures using this approach. The relation of PML to throughput measures is not so direct. This is because in previous approaches, the reward structure for a throughput measure associates rates of activity with particular states. In this paper, we choose not to formally develop the link to throughput measures further. However, the specification language will provide the necessary level of abstraction to generate throughput measures, and could do so using PML as the underlying logic. For example, with our suggested extension above, we may determine whether the throughput of our network is greater than some threshold with the expression

$$r \times \Pr((\mathtt{transmit}, r)\uparrow) > M$$

Although the logic of Huth and Kwiatkowska [12] does not support a direct interpretation of formulae as probabilities, we envisage making use of the underlying ideas to extend our approach in order to cover a wider range of equilibrium measures.

Our choice of PML was motivated by its simplicity, and its link to PEPA's strong equivalence. Other research in the area of probabilistic verification has links to our approach. Recent work by de Alfaro [19] addresses the problem of specifying "long-run" average properties of probabilistic systems. The author points out that logics such as those presented by Hansson and Jonsson [14] are able to specify bounds on probabilistic properties, but crucially these are probabilities over behaviours from a specified state. De Alfaro's approach is inspired by process algebraic tests; *experiments* are defined to represent interesting model behaviour patterns. These experiments associate a real-valued outcome with a pattern of behaviour, and are considered to occur infinitely often. The author shows how these experiments can thus be used to specify long-run behaviour. His looks to be a fruitful approach to the problem of probabilistic verification, and we too plan to focus on a concept of stochastic test. Currently work in progress, we are studying the specification of *transient* measures by considering a PEPA model in cooperation with a stochastic test [9]. However our tests do not specify long-run behaviour as do de Alfaro's experiments; rather we seek to examine the point at which a PEPA model first passes a test.

The next section demonstrates how a modeller may use the specification language, by presenting a case study of a *location tracking* system.

5 Case study: a location tracking system

As an example here we consider the problem of modelling a system where the location of people and equipment within a building is monitored by a central tracking system. Such a system is being considered for the James Clerk Maxwell Building at The University of Edinburgh. The building is notoriously confusing to navigate and the tracking system would be helpful in finding those visitors who get lost in the maze of corridors. The system would also help secretaries find professors who may be in any number of teaching and meeting rooms or colleagues' offices and would be an invaluable aid in the hunt for the (non-networked) laptop computers which can be borrowed for the secure preparation of examination papers.

Location tracking systems such as these are implemented by the use of *active badges*, credit-card sized devices which transmit unique infra-red signals which are detected by networked sensors. Systems such as these are already in use in several European universities and in research laboratories in the USA.

The University of Edinburgh issues "smart" enrolment cards at the start of each academic year. These are used both for electronic cash and as swipe-cards for door entry. It is planned that these would be superseded by cards which may also be used for location tracking. The battery life of such a device has typically been found to be around a year [20] so it is necessary to tune the performance of the system by adjusting the rate at which registration is performed in order to conserve battery power while simultaneously ensuring that the system gives accurate location information.

A Markovian stochastic process algebra such as PEPA is well suited to modelling this system because exponential registration intervals are used to prevent the repeated collisions between transmitting badges which would result in lost messages [21]. This is the same use of randomness as found in the Aloha packet-switching network: without it, a collision would inevitably be followed by another collision.

To keep the example small we will consider the simple case of tracking the progress of a single person around a single floor of a building. The floor has three corridors which are numbered 14, 15 and 16, and we assume that there is only a single sensor in each corridor. The corridors are arranged in a U-shape so that it is possible to go from the 14 corridor to the 15 corridor and then to the 16 corridor (and the other way, of course) but it is not possible to go from the 14 to the 16 corridor directly.

The behaviour of a person P who is wearing an active badge can be described in terms of their movement from one corridor to a neighbouring one and the registration of their badge with the nearest sensor.

$$
\begin{aligned}
P_{14} &\stackrel{\mathit{def}}{=} (\mathit{reg}_{14}, r).P_{14} + (\mathit{move}_{15}, m).P_{15} \\
P_{15} &\stackrel{\mathit{def}}{=} (\mathit{reg}_{15}, r).P_{15} + (\mathit{move}_{14}, m).P_{14} + (\mathit{move}_{16}, m).P_{16} \\
P_{16} &\stackrel{\mathit{def}}{=} (\mathit{reg}_{16}, r).P_{16} + (\mathit{move}_{15}, m).P_{15}
\end{aligned}
$$

Sensors accept registration information and report this back to the central database.

$$
\begin{aligned}
S_{14} &\stackrel{\mathit{def}}{=} (\mathit{reg}_{14}, \top).(\mathit{rep}_{14}, s).S_{14} \\
S_{15} &\stackrel{\mathit{def}}{=} (\mathit{reg}_{15}, \top).(\mathit{rep}_{15}, s).S_{15} \\
S_{16} &\stackrel{\mathit{def}}{=} (\mathit{reg}_{16}, \top).(\mathit{rep}_{16}, s).S_{16}
\end{aligned}
$$

For a system with only one person to be tracked the database need only store the most recently reported position.

$$
\begin{aligned}
DB_{14} &\stackrel{\mathit{def}}{=} (\mathit{rep}_{14}, \top).DB_{14} + (\mathit{rep}_{15}, \top).DB_{15} + (\mathit{rep}_{16}, \top).DB_{16} \\
DB_{15} &\stackrel{\mathit{def}}{=} (\mathit{rep}_{14}, \top).DB_{14} + (\mathit{rep}_{15}, \top).DB_{15} + (\mathit{rep}_{16}, \top).DB_{16} \\
DB_{16} &\stackrel{\mathit{def}}{=} (\mathit{rep}_{14}, \top).DB_{14} + (\mathit{rep}_{15}, \top).DB_{15} + (\mathit{rep}_{16}, \top).DB_{16}
\end{aligned}
$$

In the complete system the badge-wearer will move asynchronously but will register with the sensors. The sensors are independent but they all report back to the database. We can initialise the system in any state we wish, perhaps with the badge-wearer in the 14 corridor and the database also recording this.

$$
P \stackrel{\mathit{def}}{=} P_{14} \qquad DB \stackrel{\mathit{def}}{=} DB_{14} \qquad \mathit{System} \stackrel{\mathit{def}}{=} P \underset{\{\mathit{reg}_i\}}{\bowtie} (S_{14} \parallel S_{15} \parallel S_{16}) \underset{\{\mathit{rep}_i\}}{\bowtie} DB
$$

5.1 Analysing the performance of the location tracking system

In tuning the system to provide the best balance between accuracy and increased battery life we would investigate the probability of the database incorrectly recording the position of the badge-wearer and increase or decrease the registration rate in order to attain an acceptable threshold. Moreover, failure to register a move is not the only source of inaccurate data within the system. It is possible that reports to the database occur in the wrong order, giving a false impression of the location of the badge wearer. In either case the error can be characterised by the wearer being able to register in a location and the database not recording their presence there. If it has been decided that an acceptable level of accuracy is to have a 1% error rate then, in our high level notation, we would investigate the adequacy of our implementation as follows:

$$
\begin{aligned}
& \Pr(reg_{14}\uparrow \wedge DB \not\equiv DB_{14}) \\
+\ & \Pr(reg_{15}\uparrow \wedge DB \not\equiv DB_{15}) \\
+\ & \Pr(reg_{16}\uparrow \wedge DB \not\equiv DB_{16}) \geq 0.01
\end{aligned}
$$

We have investigated the alteration of the probability of error as the rates of badge registration and sensor reporting (variables r and s) are changed while the rate of movement of the badge-wearer (variable m) remains constant. The results are presented in Fig. 2. With the values chosen, the probability of error in the system varies between 0.02 and 0.18. These values were computed by first using the PEPA Workbench to investigate the state space of the location tracking system and then using the PEPA State Finder to select the states of interest. We then used the Maple computer algebra package both to find the steady-state probability distribution of the system and to plot the results.

As a consistency check on our work we also used the PEPA State Finder to compute the probability that the database correctly recorded the location of the badge-wearer and checked that the probabilities of database correctness and incorrectness summed to 1, which they did. The PEPA State Finder at present only implements a subset of the specification language which is described here but it is our intention to extend this to a complete implementation of the language.

The location tracking system with only a single person is a very simple system with only 72 states but we have also considered the effect of adding another person, with a correspondingly more complex database model. The state space of the system increases quickly, of course, and the extended system has 2187 states. More complex performance measures are applicable to the more complex model.

6 Further work and discussion

We are confident that the notion of probabilistic logic can be used to give a formal semantics to the specification language as described in Section 3. However, the utility of PML in interpreting transient specifications, as initially proposed in [9] has not been investigated, and remains as future work. Furthermore, we are currently developing the theory behind calculating transient measures. The satisfaction of a transient property is determined by the construction of an appropriate *stochastic test*, consisting of an ancillary PEPA process; the analysis

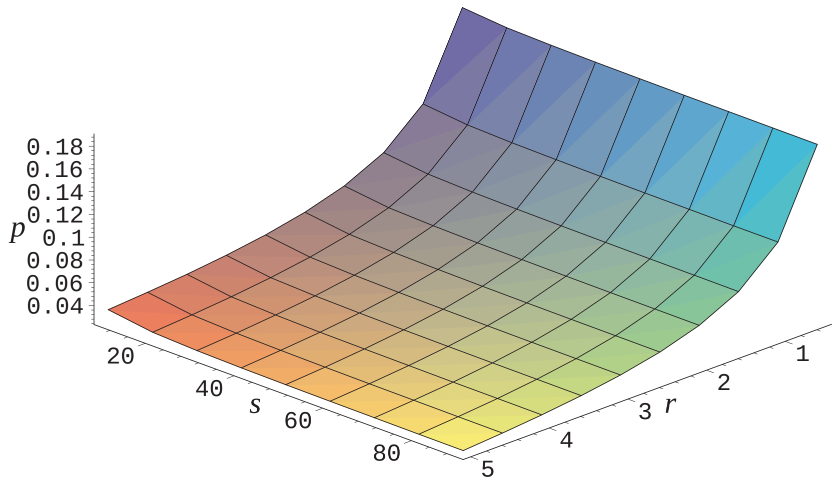

Fig. 2. Investigation of the probability p of error against rates r and s (with $m = 0.1$)

of the original model is done in concert with the stochastic test. Therefore, the longer term aim is to provide the PEPA user and modeller with a rigorous and formally defined specification language for the computation of both equilibrium and transient performance measures.

Initial work on this notation for specifying performance measures was carried out in a *feature interaction* framework. We view a *feature* as being a significant aspect or property of the system which could be used to compare this one to another. Most importantly, features are qualities which can *interact* and which can be *measured*. Therefore the feature interaction framework represents the idea that a system may be built by composing a number of well-engineered features, and is a useful paradigm for both system designers—who are concerned with a compositional or structured approach to the construction of a system—and to system users—who wish to learn and understand complex systems in terms of substantive concepts. The widespread acceptance of the importance of features makes them a very desirable concept to build into a specification language since one of the uses of a specification language can be to provide a common working language between designers and users. Our original setting meant that by using the specification language, a user could formally describe within the model a particular feature enjoyed by the system.

7 Conclusions

Despite impressive improvements in the computational power which is now available to end-users of computer systems, computer equipment remains expensive to purchase and maintain. Consequently, making cost-effective use of limited resources remains one of the motivating concerns of computer system managers. Analysis of computer systems through construction and solution of descriptive models is a hugely profitable activity: brief analysis of a model can provide as much insight as hours of simulation and measurement [22].

We have presented a notation for the description of performance specifications which relate to stochastic process algebra models expressed in the PEPA modelling notation. Our notation for performance specification focuses on models in equilibrium, and concentrates on internally measurable quantities of the system. We are currently developing the framework for specifying transient measures, which will focus on externally observable patterns in the system's transient behaviour. Further, in this paper, we have highlighted how a meaning may be given to equilibrium specifications by using the probabilistic modal logic PML. The location tracking case study illustrates the way in which a modeller would use the PEPA specification language to reason about the performance of a model.

Acknowledgements

Graham Clark and Jane Hillston are supported by the EPSRC 'COMPA' grant. Stephen Gilmore is supported by the 'Distributed Commit Protocols' grant from the EPSRC and by Esprit Working group FIREworks.

References

1. J. Hillston. *A Compositional Approach to Performance Modelling.* Cambridge University Press, 1996.
2. S. Gilmore and J. Hillston. The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling. In G. Haring and G. Kotsis, editors, *Proceedings of 7th Conf. on Mod. Techniques and Tools for Computer Perf. Eval.*, volume 794 of *LNCS*, pages 353–368, 1994.
3. 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.
4. M. Bernardo, L. Donatiello, and R. Gorrieri. Integrating Performance and Functional Analysis of Concurrent Systems with EMPA. Technical Report UBLCS-95-14, University of Bologna, 1995.
5. R. A. Howard. *Dynamic Probabilistic Systems*, volume II: Semi-Markov and Decision Processes, chapter 13, pages 851–915. John Wiley & Sons, New York, 1971.
6. H. Bowman. Analysis of a Multimedia Stream using Stochastic Process Algebra. In Priami [23], pages 51–69.
7. G. Clark. Formalising the Specification of Rewards with PEPA. In *Proceedings of the Fourth Process Algebras and Performance Modelling Workshop*, pages 139–160, July 1996.

8. G. Clark and J. Hillston. Towards Automatic Derivation of Performance Measures from PEPA Models. *Proceedings of UKPEW*, September 1996.
9. S. Gilmore and J. Hillston. Feature Interaction in PEPA. In Priami [23], pages 17–26.
10. S. Gilmore, J. Hillston, and L. Recalde. Elementary structural analysis for PEPA. Technical Report ECS-LFCS-97-377, Laboratory for Foundations of Computer Science, Department of Computer Science, The University of Edinburgh, 1997.
11. K. G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, September 1991.
12. M. Huth and M. Kwiatkowska. Quantitative analysis and model checking. In *Proceedings, Twelth Annual IEEE Symposium on Logic in Computer Science*, pages 111–122, Warsaw, Poland, 29 June–2 July 1997. IEEE Computer Society Press.
13. D. Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, 1983.
14. H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.
15. V. Hartonas-Garmhausen. *Probabilistic Symbolic Model Checking with Engineering Models and Applications*. PhD thesis, Carnegie Mellon University, 1998.
16. S. Campos, E. Clarke, and M. Minea. The Verus tool: A quantitative approach to the formal verification of real-time systems. *Lecture Notes in Computer Science*, 1254, 1997.
17. M. Bernardo. An Algebra-Based Method to Associate Rewards with EMPA Terms. In *to appear in 24th Int. Colloquium on Automata, Languages and Programming*, July 1997.
18. M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, January 1985.
19. L. de Alfaro. How to specify and verify the long-run average behavior of probabilistic systems. In *LICS: IEEE Symposium on Logic in Computer Science*, 1998.
20. A. Harter and A. Hopper. A distributed location system for the active office. *IEEE Network Magazine*, 8(1):62–70, 1994.
21. Y-B. Lin and P. Lin. Performance modeling of location tracking systems. *Mobile Computing and Communications Review*, 2(3):24–27, 1998.
22. I. Mitrani. *Probabilistic Modelling*. Cambridge University Press, 1998.
23. C. Priami, editor. *Proceedings of the Sixth International Workshop on Process Algebras and Performance Modelling*, Nice, France, September 1998.

A Proof of modal characterisation of strong equivalence

Case 1 Assume $P \cong Q$. We proceed by induction on the size of F, as in [11].

Case $F \equiv \langle \mathsf{a} \rangle_\mu G$**:** Let $P \models F$. Then by definition, there exists a set S such that $P \xrightarrow{(\mathsf{a},\nu)} S$, where $\nu \geq \mu$, and for all $P' \in S$, $P' \models G$.
Since $P \cong Q$, there exists some strong equivalence $\mathcal{R}$, such that for all $a \in \mathcal{A}$, for all $S \in \mathcal{C}/\mathcal{R}, q[P, S, \mathsf{a}] = q[Q, S, \mathsf{a}]$. For each $P' \in S$, let $\mathcal{R}_{P'}$ be the equivalence class in $\mathcal{C}/\mathcal{R}$ which contains P'. Furthermore, let $S'' = \bigcup_{P' \in S} \mathcal{R}_{P'}$. Now, for each $P'' \in S''$, $P''\mathcal{R}P'$, and thus $P'' \cong P'$, for some $P' \in S$, and so by the hypothesis, for all $P'' \in S''$, $P'' \models G$.

Since $S \subseteq S''$, $P \xrightarrow{(\mathsf{a},\nu')} S''$, where $\nu' \geq \nu$. Since $P\mathcal{R}Q$, for all $T \in \mathcal{C}/\mathcal{R}$, $q[P, T, \mathsf{a}] = q[Q, T, \mathsf{a}]$. However, note that for all $s, s' \in S$, $\mathcal{R}_s = \mathcal{R}'_s$ or $\mathcal{R}_s \cap \mathcal{R}_{s'} = \emptyset$. Therefore, by construction of S'', $q[P, S'', \mathsf{a}] = q[Q, S'', \mathsf{a}]$.

Therefore, $Q \xrightarrow{(\mathsf{a},\nu')} S''$, where $\nu' \geq \nu \geq \mu$; and for all $Q' \in S''$, $Q' \models G$. Therefore, $Q \models \langle \mathsf{a} \rangle_\mu F$. By symmetry of $\cong$, this case is complete.

Case $F \equiv \nabla_{\mathsf{a}}$**:** Let $P \models F$. Therefore $P \overset{\mathsf{a}}{\not\rightarrow}$. Since $P \cong Q$ it is the case that for some strong equivalence $\mathcal{R}$, for all $a \in \mathcal{A}$, for all $S \in \mathcal{C}/\mathcal{R}, q[P, S, \mathsf{a}] = q[Q, S, \mathsf{a}]$. However, for all $S' \subseteq 2^{\mathcal{C}}$, $q[P, S', \mathsf{a}] = 0$. Since for all $S \in \mathcal{C}/\mathcal{R}$, $S \subseteq 2^{\mathcal{C}}$, it is the case that $q[Q, S, \mathsf{a}] = q[P, S, \mathsf{a}] = 0$ for any $S \in \mathcal{C}/\mathcal{R}$, for any $\mathcal{R}$ which is a strong equivalence. Therefore there does not exist a C such that $q[Q, C, \mathsf{a}] > 0$ and therefore, $Q \overset{\mathsf{a}}{\not\rightarrow}$. By symmetry of $\cong$, this case is complete. All other cases are straightforward.

Case 2 Assume that for all F, $P \models F$ if and only if $Q \models F$. Let $\mathcal{R} = \{(P, Q) \mid$ for all F, $P \models F$ if and only if $Q \models F\}$. The result will hold if $\mathcal{R}$ can be shown to be a strong equivalence. By simple inspection, $\mathcal{R}$ is clearly an equivalence relation. Thus, it must be shown that for all $\mathsf{a} \in \mathcal{A}$, for all $S \in \mathcal{C}/\mathcal{R}$, $q[P, S, \mathsf{a}] = q[Q, S, \mathsf{a}]$.

Let $S \in \mathcal{C}/\mathcal{R}$. Assume that for some $a \in \mathcal{A}$, $P \xrightarrow{(\mathsf{a},\mu)} S$ for some μ. Now consider the a-derivatives of Q, labelled $Q'_1, \ldots, Q'_m, Q'_{m+1}, \ldots, Q'_n$, where $n \geq 0$, $0 \leq m \leq n$. These derivatives are labelled such that $Q'_1, \ldots, Q'_m \in S$ and $Q'_{m+1}, \ldots, Q'_n \notin S$. For each Q'_i, let the rate at which Q makes an a-transition to Q'_i be denoted by μ_i. Since S is an equivalence class under $\mathcal{R}$, it is the case that for each $Q'_j, m+1 \leq j \leq n$, there exists a formula F'_j such that $Q'_j \models F'_j$, and for each $Q'_i, 0 \leq i \leq m$, $Q'_i \not\models F'_j$. From a lemma by Larsen and Skou [11], it is possible to construct a dual formula to each F'_j, named here $\overline{F}'_j$ such that for $Q'_j, m+1 \leq j \leq n$, $Q'_j \models F'_j$ if and only if $Q'_j \not\models \overline{F}'_j$. Now it is the case that $Q \models \langle a \rangle_{\mu'} (\overline{F}'_{m+1} \wedge \ldots \wedge \overline{F}'_n)$, where $\mu' = \sum_{i=1}^{m} \mu_i$. However, by the initial assumption, it is also the case that $P \models \langle a \rangle_{\mu'} (\overline{F}'_{m+1} \wedge \ldots \wedge \overline{F}'_n)$, and therefore, $\mu' \geq \mu$. This then gives that $q[Q, S, \mathsf{a}] = \mu' \geq \mu$. However, $\mathcal{R}$ is symmetrical, and so by such an argument, it is the case that $\mu \geq \mu'$, and thus that $\mu = \mu'$. Hence $q[P, S, \mathsf{a}] = q[Q, S, \mathsf{a}]$, as required. $\square$