Terminating processes in PEPA

Nigel Thomas* and Jeremy Bradley†

Research Institute in Software Evolution
University of Durham, UK

Abstract

In this paper we introduce the concept of a null component into the Markovian process algebra PEPA. The null component allows us to explicitly define models which have components that cease to execute. By implication this also allows us to formally define a notion of a successfully terminating model and hence enhanced concepts of deadlock and livelock in transient PEPA models.

1 Introduction

Stochastic process algebras, such as PEPA [3] have been used for some time as a formal mechanism for specifying and manipulating performance models in a compositional way. The syntax of PEPA is deliberately small, and yet it provides a very large degree of expressibility of Markovian behaviour. A number of powerful techniques have been developed which allow complex models to be simplified and solved in an efficient manner, however, until recently, implementations of tools supporting PEPA and other stochastic process algebra (SPA) have concentrated on being able to solve models to find steady state results. There are many reasons why this has been so, not least being that steady state results form the basis for the majority of performance metrics commonly derived. In addition, the primary motivation for developing PEPA and other SPAs has been to exploit the advantages of a formal compositional approach not available in other modelling paradigms. Other tools supporting other paradigms, in particular stochastic Petri nets, have supported transient analysis, although often using simulation.

More recently Bradley et al [5, 6] have developed some analytic solution methodology for transient properties of semi-Markov processes and SPA. This method also applies to Markov processes and a tool has been developed to derive transient solutions from PEPA models. In extending the range

*nigel.thomas@durham.ac.uk
†jeremy.bradley@durham.ac.uk

of solutions that can be derived directly this tool has also extended the range of model that can be adequately studied. In the PEPA Workbench and other existing PEPA tools [2, 4] only cyclic PEPA models [1] are considered for the obvious reason that only cyclic models will give a meaningful steady state solution across the entire state space. However, in considering the transient behaviour of models this restriction is not necessary and we may derive solutions for models with absorbing states or reduced sets of behaviours.

In this paper we consider some options for defining models with non-cyclic properties, using just the existing PEPA syntax or different interpretations of a simple extension. We concentrate on the concept of a *Stop* component, i.e. a component that has no future behaviour. We begin the paper by re-introducing Hilston's Markovian process algebra, PEPA [3]. In Section 3 we introduce the concept of a *Stop* component into the syntax of PEPA and we compare this notion of null behaviour with other possibilities. In Section 4 we explore some of the useful implications of using the null component for both specifying and analysing transient models and present some examples. In Section 5 we present a larger example and show how the *Stop* component notion supports transient analysis. In Section 6 we give some conclusions and future directions of research.

2 PEPA

A formal presentation of PEPA is given in [3], in this section a brief informal summary is presented. PEPA, being a Markovian Process Algebra, only supports actions that occur with rates that are negative exponentially distributed. Specifications written in PEPA represent Markov processes and can be mapped to a continuous time Markov chain (CTMC). Systems are specified in PEPA in terms of the interaction of *components* over local or shared *activities*. An activity (α, r) is described by the *type* of the activity, α, and the *rate* of the associated negative exponential distribution, r. This rate may be any positive real number, or given as *unspecified* using the symbol $\top$. The syntax for describing components is given as:

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

The component $(\alpha, r).P$ performs the activity of type α at rate r and then behaves like P. The component $P + Q$ behaves either like P or like Q, the resultant behaviour being given by the first activity to complete.

The component P/L behaves exactly like P except that the activities in the set L are concealed, their type is not visible and instead appears as the unknown type τ.

Concurrent components can be synchronised, $P \underset{L}{\bowtie} Q$, such that activities in the *cooperation set* L involve the participation of both components. In PEPA the shared activity occurs at the slowest of the rates of the participants and if a rate is unspecified in a component, the component is passive

[1] in essence every state in the CTMC is reachable from every other state

with respect to that activities of that type. The parallel combinator $\|$ is used as shorthand to denote synchronisation with no shared activities, i.e. $P \| Q \equiv P \bowtie_{\emptyset} Q$. $A \stackrel{def}{=} P$ gives the constant A the behaviour of the component P.

The *derivative* of a component is defined formally in [3], informally a derivative is the "state" of a component defining its current behaviour. For example, if $P \stackrel{def}{=} (\alpha, r).Q$ then Q is a derivative of P and if $Q \stackrel{def}{=} (\beta, r_2).R$ then R is a derivative of both Q and P, and so on. The derivative set, $ds(P)$, is the set of all the possible derivatives of a component, P.

2.1 Model Analysis

Traditionally in performance modelling we have primarily focussed on deriving a steady state solution of the global state space and deriving metrics from this. This is the only form of analysis explicitly supported by the PEPA Workbench [2], although other forms of numerical solution are possible through manipulation of the generator matrix in Maple or Matlab. The conditions for the existence of a steady state solution for a continuous time Markov chain are well known, for a finite Markov chain, it is sufficient that the chain is irreducible[2] (every state is reachable from every other state). Clearly, working at the component level as in PEPA, we do not wish to form the global state space only to find that a steady state solution does not exist, hence the notion of *cyclic PEPA* has been formulated [3] to give some preconditions on the formation of PEPA components and interactions between them. Part of this process is the separation of PEPA operators to give *sequential* and *model* components:

$$\begin{array}{rl} \textit{sequential components} & S ::= (\alpha, r).S \mid S + S \mid A \\ \textit{model components} & P ::= S \mid P \bowtie_{L} P \mid P/L \end{array}$$

In cyclic PEPA it is insisted that every sequential component, S, is in itself *cyclic*, that is, its derivative set, $ds(S)$, does not change with time, i.e. $ds(S') = ds(S) \; \forall \; S' \in ds(S)$. Clearly a parallel combination of cyclic sequential components will give rise to an irreducible CTMC.

3 Null behaviour in PEPA

We now wish to explore the possibility that a component does not continue indefinitely, but instead at some point it reaches the conclusion of its activities. It is possible to define such a scenario

[2]This condition is clearly not sufficient for infinite Markov chains, where knowledge of the relative values of rates is required, however the PEPA Workbench only supports finite state models.

within the existing syntax of PEPA by considering (conceptually) null behaviour:

$$\begin{aligned} S &\stackrel{def}{=} (\alpha, r).Finish \\ Finish &\stackrel{def}{=} (null, x).Finish \end{aligned}$$

In this example the component P will "finish" after completing its action α, however it will not actually terminate, instead it will perform a *null* action ceaselessly. In practice the rate of the *null* action is immaterial since no other action will be performed by this component, we could chose to leave the rate unspecified i.e. $\top$, or to set it to zero, meaning the *null* action will never actually occur.

Instead of this somewhat oportunisitic approach, we have chosen instead to explicitly define the termination of a component behaviour. We add to the simple set of operators a reserved behaviour which we refer to as *Stop*. If $P \stackrel{def}{=} (\alpha, r).Stop$ then the component P will perform the action α at rate r and then stop. We also can give specific names to stopped components by the conventional mechanism, e.g.

$$\begin{aligned} S &\stackrel{def}{=} (work, r).S + (breakdown, f).Fail \\ Fail &\stackrel{def}{=} Stop \end{aligned}$$

Once a component enters a stop behaviour it ceases to undertake any actions and it cannot be restarted, this has several implications for shared actions. Clearly, in the simple sequential case, $Stop + P \equiv P$ and $Stop + Stop \equiv Stop$, however we have several options as to how to interpret $P \underset{L}{\bowtie} Stop$. A simple (simplistic) approach might be to say that $P \underset{L}{\bowtie} Stop \equiv Stop$, that is, if one component in a system stops, the whole system stops. Clearly there are real systems where such behaviour can occur, however despite the attraction of simplicity this definition is not consistent with the loose coupling principal, i.e. components should not have to know the states of other components. The next obvious possibility is for stopped components to be ignored, i.e. $P \underset{L}{\bowtie} Stop \equiv P$. If we allow the system to proceed without considering the stopped component then we are allowing shared actions to proceed with just one participating component. Clearly we could allow this, especially if no action $\alpha \in L$ is passive in P. Again this is attractively simple, however it ignores the cooperative nature of shared actions; if the arrival process stops a queue cannot continue to accept jobs because there are no jobs to accept. Instead we chose a slightly more complex definition, whereby $P \underset{L}{\bowtie} Stop \equiv P^{-L}$ and P^{-L} is the component P with all actions $\alpha \in L$ blocked. This definition not only allows components to proceed independently when so specified, but also reflects the cooperative nature of shared actions.

4 Properties of terminating processes in PEPA

On first appearances we have achieved very little through this definition of a *Stop* component. Consider, for example, the following Markov chains:

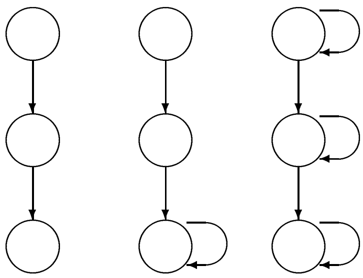

Figure 1. Three similar Markov chains

The left most chain represents a transient model where all behaviour ceases when the last state is reached. The middle chain represents the same sequence of be behaviours with an absorbing behaviour in the last state. The right hand chain represents the case where recurrent behaviours are present in every state. If the transitions between the states are the same for each of the three chains and all transitions are Markovian then the transient solution for each chain will be identical. Given this fact then there is no advantage in our *Stop* component over the existing mechanisms as illustrated by the *Finish* component in the earlier example.

In fact, the main advantage is in making *Stop* a reserved component with an explicit meaning. This means that the component with a statement $Fail \stackrel{def}{=} Stop$ is explicitly a transient component amenable only to transient analysis, whereas the component with the statement $Finish \stackrel{def}{=} (null, x).Finish$ is not explicitly transient and may be analysed in steady state. More importantly the *Stop* component allows us to define properties of a transient process.

Clearly sequential components, S and T, will be terminating components if $Stop \in ds(S)$ and $Stop \in ds(T)$, however a model component $P \stackrel{def}{=} S \underset{L}{\bowtie} T$ may not necessarily terminate, indeed it may be irreducible! Consider the following simple (pathological) example.

$$
\begin{aligned}
S &\stackrel{def}{=} (\alpha, r1).S + (failS, fs).Stop \\
T &\stackrel{def}{=} (\beta, r2).T + (failT, ft).Stop \\
P &\stackrel{def}{=} S \underset{\{failS, failT\}}{\bowtie} T
\end{aligned}
$$

Clearly neither S or T will ever stop since the actions *failS* and *failT* cannot happen. Obviously we would not normally define a model to have behaviours that intentionally never happen, thus if we were to construct an irreducible CTMC from a model defined with *Stop* components it would probably indicate an error in the model specification and as such should be flagged in model checking. In fact we can go further than this and introduce the familiar notions of *deadlock* and *livelock* in the context of transient PEPA models.

By strict definition a model $P \stackrel{def}{=} Stop||Stop$ is deadlocked, i.e. it cannot proceed. Of course, in this case all components have reached a conclusion and this is the desirable outcome for any transient model. This then gives us a notion of *correct termination* for the process, in fact we can say a transient model is correct if, and only if, in all absorbing nodes of its derivation graph every component is equivalent to *Stop*. If there exists an absorbing node in the derivation graph where this is not true then this must indicate possible deadlock, i.e. the model has executed a path whereby no possible sequence of actions will allow all its components to stop, instead one or more components must be blocked on a shared action. Consider the following two simple examples.

$$\begin{aligned} S1 &\stackrel{def}{=} (\alpha, r1).(failS, fs).Stop \\ T1 &\stackrel{def}{=} (\alpha, r2).(failT, ft).Stop \\ P1 &\stackrel{def}{=} S1 \underset{\alpha}{\bowtie} T1 \end{aligned}$$

$$\begin{aligned} S2 &\stackrel{def}{=} (\alpha, r1).S2 + (failS, fs).Stop \\ T2 &\stackrel{def}{=} (\alpha, r2).(failT, ft).Stop \\ P2 &\stackrel{def}{=} S2 \underset{\alpha}{\bowtie} T2 \end{aligned}$$

In the first case the two components share an action α and then independently each perform an action leading to *Stop*. In the second the same path is also possible, so there is a correct termination, but it is also possible that $S2$ performs the action *failS* before α, resulting in $S2$ becoming *Stop* and $T2$ being blocked waiting on α. The first model has guaranteed correct termination, the second may deadlock.

Finally, if a component may continue to perform some actions but cannot stop and the model has a reducible CTMC then the model is said to have livelock. For example:

$$\begin{aligned} S3 &\stackrel{def}{=} (\alpha, r1).S3 + (failS, fs).Stop \\ T3 &\stackrel{def}{=} (\beta, r3).T3 + (\alpha, r2).(failT, ft).Stop \\ P3 &\stackrel{def}{=} S3 \underset{\alpha}{\bowtie} T3 \end{aligned}$$

In this case the component $T3$ will always be able to perform the action β and if $S3$ performs *failS* before α then $T3$ will never stop.

4.1 Database Example

We illustrate the use of terminating components in a PEPA-like language by analysing a face-recognition database. Here the database can fail to find any match on a given face, or if found a match might be false or true, hence the operator gets to decide whether the result is correct or not. Thus neither *operator* or *dbase* is wholly a slave or master component, as can be seen in the process algebra description in Figure 2.

$$
\begin{aligned}
operator &\stackrel{def}{=} (analyse\_init, \top).operator_1 \\
operator_1 &\stackrel{def}{=} (return\_data, \top).operator_2 + (no\_data\_left, \top).\mathbf{0}_{\mathbf{none}} \\
operator_2 &\stackrel{def}{=} (correct\_data, \nu_c).\mathbf{0}_{\mathbf{fnd}} + (incorrect\_data, \nu_i).\mathbf{0}_{\mathbf{notfnd}} \\
\mathbf{0}_{\mathbf{none}} &\stackrel{def}{=} Stop \\
\mathbf{0}_{\mathbf{fnd}} &\stackrel{def}{=} Stop \\
\mathbf{0}_{\mathbf{notfnd}} &\stackrel{def}{=} Stop \\
\\
dbase &\stackrel{def}{=} (analyse\_init, \lambda_0).dbase_1 \\
dbase_1 &\stackrel{def}{=} (query\_database, \lambda_1).dbase_2 \\
dbase_2 &\stackrel{def}{=} (analyse\_result, \lambda_0).dbase_3 \\
dbase_3 &\stackrel{def}{=} (compare, \lambda_2).dbase_4 \\
dbase_4 &\stackrel{def}{=} (found\_data, \mu_0).dbase_5 + (fail, \mu_1).dbase_1 \\
&\quad + (no\_data\_left, \mu_2).dbase_r \\
dbase_5 &\stackrel{def}{=} (return\_data, \alpha_1).dbase_r \\
dbase_r &\stackrel{def}{=} Stop \\
\\
system &\stackrel{def}{=} operator \underset{L}{\bowtie} dbase \\
L &= \{analyse\_init, return\_data, no\_data\_left\}
\end{aligned}
$$

Figure 2. A database analysis by MPA with null states.

The important issue to highlight concerning the system is that both *dbase* (Figure 4) and *operator* (Figure 3) are terminating components, the overall state space is dominated by the *Stop* states and is not irreducible, as shown in Figure 5.

Clearly this model conforms to the notion of correct termination as introduced in the previous section and there is important performance information to be derived. Traditionally this would not

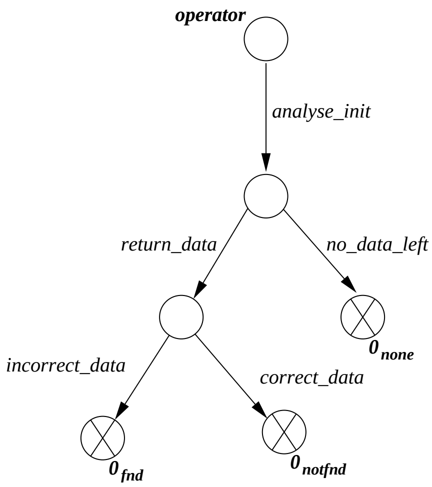

Figure 3. The component state space, *operator*.

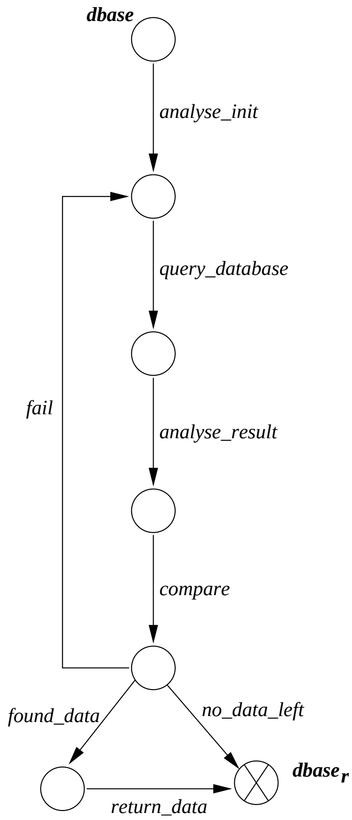

Figure 4. The component state space, *dbase*.

be an analysable system, as irreducibility is a requirement of steady-state analysis. However with

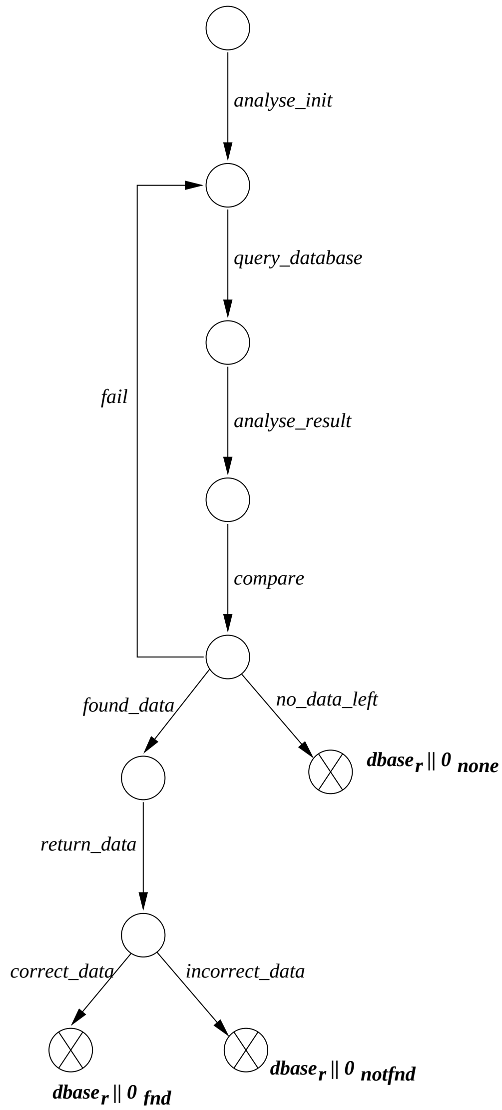

Figure 5. The global state space, $operator \underset{L}{\bowtie} dbase$.

the advent of stochastic aggregation techniques and improved numerical methods for inverting Laplace transforms we can extract transient absorption probabilities for ending up in one of the three null states.

We define $p_{\mathrm{x}}(t) = \mathbb{P}(\text{system is in state } \mathbf{0_x} \text{ at time } t)$. and present the three quantities $p_{\mathrm{none}}(t)$, $p_{\mathrm{notfnd}}(t)$, $p_{\mathrm{fnd}}(t)$ in Figure 6 and the total transient absorption probability in Figure 7. To do this, we specify typical values for λ_i, μ_j, and ν_k so as to extract specific functions for a given instance of the database.

Using stochastic aggregation over the three possible paths of the system, we obtain the following analytic transforms for our three required path distributions, $q_{\text{fnd}}(t)$, $q_{\text{nofnd}}(t)$ and $q_{\text{none}}(t)$:

$$\hat{q}_{\text{fnd}}(z) = \frac{\lambda_0}{\lambda_0 + z} \frac{\mu_0}{\mu_0 + \mu_1 + \mu_2 + z} \frac{\nu_c}{\nu_c + \nu_i + z} \Psi(z) \tag{4.1}$$

$$\hat{q}_{\text{notfnd}}(z) = \frac{\lambda_0}{\lambda_0 + z} \frac{\mu_0}{\mu_0 + \mu_1 + \mu_2 + z} \frac{\nu_i}{\nu_c + \nu_i + z} \Psi(z) \tag{4.2}$$

$$\hat{q}_{\text{none}}(z) = \frac{\lambda_0}{\lambda_0 + z} \frac{\mu_2}{\mu_0 + \mu_1 + \mu_2 + z} \Psi(z) \tag{4.3}$$

where

$$\Psi(z) = \frac{\Pi(z)}{1 - \frac{\mu_1}{\mu_0 + \mu_1 + \mu_2 + z} \Pi(z)} \tag{4.4}$$

$$\Pi(z) = \frac{\lambda_0}{\lambda_0 + z} \frac{\lambda_1}{\lambda_1 + z} \frac{\lambda_2}{\lambda_2 + z} \tag{4.5}$$

Now we note that $p_{\text{x}}(t)$, our desired metric, is the cumulative distribution function of the equivalent path distribution $q_{\text{x}}(t)$. Therefore, we are able to use the fact that $\hat{p}_{\text{x}}(z) = \hat{q}_{\text{x}}(z)/z$ and invert the Laplace function required to obtain the cumulative distribution function.

We now set $\lambda_0 = 1.0$, $\lambda_1 = 3.2$, $\lambda_2 = 2.6$, $\mu_0 = 0.2$, $\mu_1 = 5.7$, $\mu_2 = 0.05$, $\nu_c = 2.0$ and $\nu_i = 0.5$ to obtain the Laplace inversions shown in Figures 6, and 7.

5 Conclusions and further work

In this paper we have introduced a very minor extension to PEPA but one that is both extremely useful and has significant benefits in defining properties of transient PEPA models. The discussion on properties of transient models we present here is hardly new in the wider context of process algebra. Hoare [1] has a very similar concept of a *Stop* behaviour, although the exact interpretation with respect to cooperation is obviously different in that case. Furthermore, we have also been able to show how this extension supports the solution of transient models and the derivation of transient measures important for performance modelling. Again, this solution is not difficult or especially novel, but the duality of the link between specification and analytical derivation of the measure is significant. As such this paper is very much in the spirit of process algebra developments for performance modelling in combining the ability to reason about properties of a model and analytically derive solutions based on its stochastic behaviour.

Some more work remains to be completed in this area. In particular we have not presented here the operational semantics of the *Stop* component which are need if others are to follow our approach in implementation (although the concept should be simple enough to follow without).

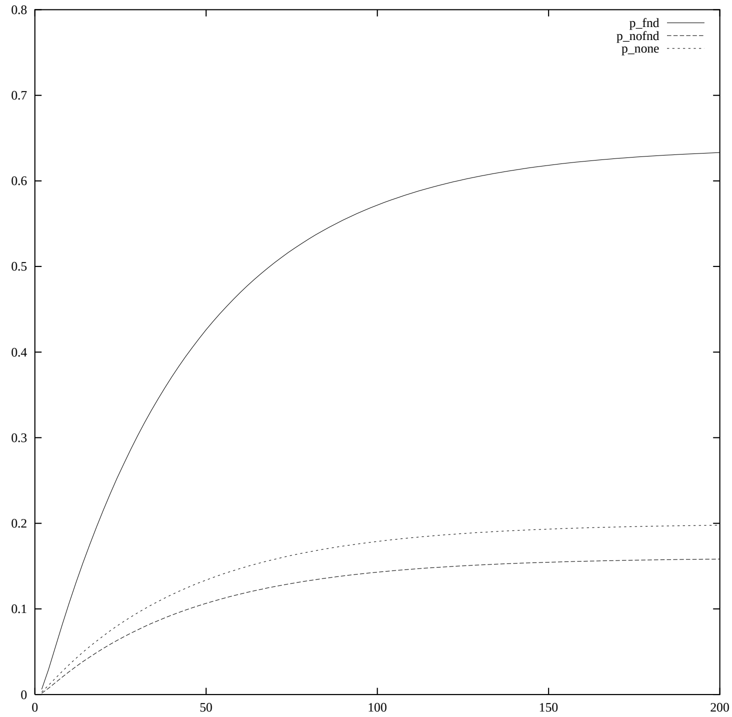

Figure 6. The transient absorption probabilities, $p_{\mathrm{fnd}}(t)$, $p_{\mathrm{notfnd}}(t)$ and $p_{\mathrm{none}}(t)$.

Acknowledgements

The authors are supported by the EPSRC funded QWiD project (reference GR/N01491) further details of which are available at
http://www.dur.ac.uk/nigel.thomas/qwid.html.

References

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

[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 eds., *Proceedings of the Seventh International Conference on Modelling Techniques and Tools for Computer Performance Evaluation,* Lecture Notes in Computer Science, vol. 794, pp. 353–368, Springer Verlag, 1994.

[3] J. Hillston, *A Compositional Approach to Performance Modelling*, Cambridge University Press, 1996.

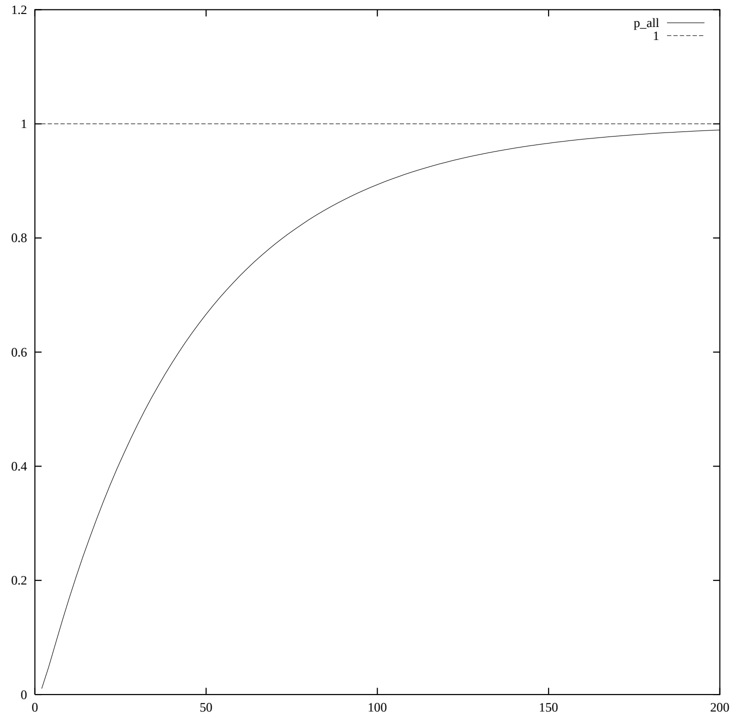

Figure 7. The total transient absorption probability.

[4] G. Clark, S. Gilmore, J. Hillston and N. Thomas, Experiences with the PEPA performance modelling tools, *IEE Proceedings - Software*, **146**(1), 1999.

[5] Jeremy Bradley and Neil Davies, A Matrixbased Method for Analysing Stochastic Process Algebras, in: Proceedings of the 8th International Workshop on Process Algebra and Performance Modelling, Carleton Scientific Press, 2000.

[6] J. Bradley and N. Thomas, Extracting transient functions from SPA models, *submitted for publication.*