Rapid performance evaluation using fluid-flow analysis and eXtended Stochastic Probes

Allan Clark and Stephen Gilmore

Laboratory for Foundations of Computer Science
The University of Edinburgh

Abstract. Rapid and accessible performance evaluation of complex software systems requires two critical features: first, the ability to specify useful performance metrics easily; second, the ability to analyse massive state spaces of 10^{1000} states and beyond. In this paper we combine previous work on the development of stochastic probes as a performance query specification language with the requirement to analyse massive state spaces in order to obtain a response-time profile. A stochastic probe is a passive component added to the model in order to partition the states of the model for the purpose of analysing a particular query. The language of stochastic probes is a specification language with a regular-expression like syntax which allows the concise specification of a sequence of activity observations. The specification is then automatically translated into a passive component which is inserted into the model. The language of "eXtended Stochastic Probes" (XSP [7]) allows the modeller to specify guards on activity observations. We have found stochastic probes to be a convenient query specification mechanism which is particularly well-suited to passage-time queries in which the user must specify source and target events. An event is either the occurrence of a particular activity or some condition becoming true. In this paper we restrict ourselves to a subset of all passage-time queries called a response-time query in which the source and target events are observable by one of many components of the same kind.

1 Introduction

Performance modelling techniques are used to build abstract models of a real-world system. The 'real-world system' may not yet exist, or – if it does – we may need to change it, or the environment which surrounds it may change outside of our control. Dynamic stochastic modelling is often used to predict the performance of a combination of system and environment without the cost of deploying a real-world test, which may be infeasible or prohibitively expensive. Models are analysed to give performance results which are applicable to the real-world system. Stochastic process algebras are a popular method of building these abstract models for many reasons. Process algebras allow large models to be built from small components which are combined and replicated. As a result, models can be created which would be infeasible to construct in a less compositional manner. Additionally, the precise formal definition of stochastic process algebras allows static analysis to be performed over the model. This can prevent many modelling errors.

Performance models, whether described in a process algebra such as PEPA [16] or a graphical notation such as Petri nets, are often translated into a continuous-time Markov chain (CTMC) for analysis. This approach to evaluating the model suffers from the well-known problem of state-space explosion. As we increase the populations of the components involved in the model – for example increasing the number of *user* or *client* components – the size of the resulting state-space grows exponentially. Even utilising techniques to dampen the effect of state-space explosion, such as aggregation [24, 11], or using simulation to approximately model-check the CTMC [21], the rapid grow of the discrete state-space as components are added greatly limits the applicability of this approach to analysing a model. This problem hinders modelling to the extent that in recent years modellers have often turned to alternative approaches to model analysis which do not require the generation of the entire state-space.

For the PEPA stochastic process algebra the breakthrough for this style of analysis was Hillston's 2005 paper[17] which describes the automatic translation of a PEPA model into a set of coupled ordinary differential equations (ODEs). The number of equations produced from the translation depends on the number of distinct component states and not the populations of the component types. This means that users of the PEPA language are able to analyse models that – if interpreted in the discrete setting – would result in state-space sizes of the order of 10^{1000} and beyond. When a PEPA model is translated into a set of ODEs this is often referred to as the fluid-flow approximation of the model. Since then, the Stochastic Simulation Algorithm(SSA) [10] has also been used [2] to analyse large-scale PEPA models.

The advantage of using such techniques is clear. Models which were previously infeasible to analyse (using CTMCs) now fall within the realm of models appropriate for analysis via description in PEPA. The disadvantage is the loss of the vast body of knowledge of analysis techniques for CTMCs.

One kind of analysis which can be achieved in various ways including uniformisation [12, 13, 23, 4] (also known as randomisation), is the extraction of *passage-time* quantiles [9, 14]. This allows the calculation of the probability of moving from one set of source states to another set of target states at or within a given time after a source state is entered. In the world of the CTMC the source and target of a passage are both defined as a set of states. However, when specifying the query we have not yet derived the CTMC, hence we talk about the source and target events. A source event is either an activity which results in a transition into a source state, or a condition which becomes true when such a transition occurs. A target event is analogously defined.

In this paper we consider a subset of all passage-time queries which we call *response-time* queries resulting in a *response-time profile*. This subset consists of all passage-time queries which measure the probability of completing a passage between two events observable by a single component. This is often a single component actively initiating a request activity and then waiting to passively or actively cooperate in a response activity. Although we call this a response-time profile the initiating and completing activities can conceptually have nothing to do with a request and response. However, for the remainder of this paper, we will refer to them as request and response activities.

A response-time profile is a more detailed report of the responsiveness of a system than an average response-time. A response-time profile reports the probability of a component observing the completion of a response at a given time after the initiation of the request. For large-scale systems which cannot be compiled to a CTMC we have previously been limited to the computation of average response-times. Numerical integration of the derived system of ODEs is used to predict the evolution of the population of each component type over time within the system. Where this converges to an unchanging equilibrium in which the population of each component type remains the same we can classify this as the steady-state of the system. Using this steady-state and an application of *Little's Law* [22] we can extract average response-times [6]. The main contribution of this paper is the generalisation of this technique into one for obtaining a full response-time profile based on the fluid approximation.

Structure of this paper: The remainder of this paper is structured as follows; the following sub-section provides a brief overview of PEPA, the stochastic process algebra with which this paper is concerned. Following this, Section 2 recaps in detail how stochastic probes allow the calculation of average response-times from PEPA models which we translate into ODEs. Section 3 shows how we can obtain response-time profiles by modifying the way in which the probe specification is translated. This is important as it means that the user need not change their measurement specification and the calculation of the response-time profile can be entirely automated. We give two examples and validate these by comparing the results with those obtained using a stochastic simulator. In addition we build a more abstract model with fewer states such that it may be translated into a CTMC and compare the results of all three techniques. In this section the queries are kept simple in that each response-passage has only one source state; this simplifies the calculation a little for the purposes of demonstration. However in Section 4 the method is generalised by removing this restriction and allowing for response-passages which have multiple source states. Finally we conclude on our method in Section 5.

1.1 PEPA

We work with the Markovian process algebra PEPA, as defined in [16]. Applications of the language are described in [15, 19, 18, 1]. PEPA is a stochastically-timed process algebra where sequential components are defined using prefix and choice. Models compose these sequential components, requiring them to cooperate on some activities, and hide others. Rates are associated with activities performed by each component and the passive rate $\top$ is used to indicate that the component will passively cooperate with another component on this activity. In this case the passive component may enable or restrict the activity from being performed by the cooperating component but the rate when enabled is determined by the actively cooperating component. The component $(a, r).P$ performs the activity a at rate r whenever it is not blocked by a cooperating component and becomes the process P. The component $(a, \top).Q$ passively synchronises on the activity a and becomes process Q. We use the version of PEPA with arrays of components and functional rates [20] ("marking dependent rates", in Petri nets terms). We write $P[5]$ to denote five copies of the component P which do not cooperate and $P[5][\alpha]$ to denote five copies of the component P which cooperate on the activity α. That is, $P[5]$ is an abbreviation for $P \parallel P \parallel P \parallel P \parallel P$ and $P[5][\alpha]$ is an abbreviation for $P \underset{\{\alpha\}}{\bowtie} P \underset{\{\alpha\}}{\bowtie} P \underset{\{\alpha\}}{\bowtie} P \underset{\{\alpha\}}{\bowtie} P$.

We use the notation $P \underset{*}{\bowtie} Q$ to be a synonym for $P \underset{\mathcal{L}}{\bowtie} Q$ where $\mathcal{L}$ is the set of activities performed by both P and Q, i.e. the intersection of their alphabets. The component *Stop* indicates a component which has terminated and can no longer perform any activities. Finally our probe language makes use of immediate actions which are written $a.P$ to mean the process which instantaneously performs the action a to become the process P. These are generally cooperated over such that components can be blocked until another component has entered a state which may perform the appropriate immediate synchronisation.

A PEPA model can be compiled into several different formats for analysis. There are three techniques commonly used to analyse a PEPA model; translation to a continuous time Markov chain (CTMC) [16], or to a set of ordinary differential equations (ODEs) [17], and the use of stochastic simulation [2].

2 Average Response-Time

In this section we detail how average response-time can be calculated for a model via translation to a system of ODEs.

The model shown in Figure 1 represents an e-commerce web-site. There are a number of user components all of which must make requests to a central *Server* component. This may in reality be many physical servers but is represented here as a single stateless component. Each *User* may, independently, make a single *browse* or *buy* request corresponding to the user clicking on a link to view or purchase an item. The *Server* component then responds after some delay to the user with either a page or a confirmation of payment. The rate of responses to *buy* requests is somewhat slower since it takes longer to process a purchase request than a data request.

We can obtain the average-response times of both the *browse* and *buy* request activities using this model by solving for the long-term, or steady-state, populations of the three user component states. Where $Pop_{st}(P)$ means the steady-state population of the state P and $Thr(\alpha)$ means the throughput (at steady-state) of the action α, then our average response-times are computed as:

$$\begin{aligned} AvgRes(browse) &= \frac{Pop_{st}(Browse)}{Thr(browse)} \\ &= \frac{Pop_{st}(Browse)}{Pop_{st}(User) \times \lambda_{br}} \end{aligned}$$

$$\begin{aligned}
User &\overset{def}{=} (browse, \lambda_{br}).Browse + (buy, \lambda_{br}).Buy \\
Browse &\overset{def}{=} (getPage, \lambda_{gp}).User \\
Buy &\overset{def}{=} (getConfirm, \lambda_{gc}).User \\
\\
Server &\overset{def}{=} (getPage, \lambda_{sp}).Server + (getConfirm, \lambda_{sc}).Server \\
\\
System &\overset{def}{=} (Server[M]) \underset{\mathcal{L}}{\bowtie} (User[N]) \\
&\quad \text{where } \mathcal{L} = \{\, getPage, getConfirm \,\}
\end{aligned}$$

Fig. 1. The PEPA model for an e-commerce web-site

$$\begin{aligned}
AvgRes(buy) &= \frac{Pop_{st}(Buy)}{Thr(buy)} \\
&= \frac{Pop_{st}(Buy)}{Pop_{st}(User) \times \lambda_{bu}}
\end{aligned}$$

The calculation of these particular response-times is made straightforward by the fact that there is a set of states of the user components which corresponds to the user being in the middle of the analysed response passage. In these specific cases we are analysing single action response times and the sets of states are singleton sets. However if we wish to measure the response-time from an initial *browse* until a confirmation of a payment then we must modify our model. This is because there is no set of $User$ states which indicate that the user has performed an initial *browse* request but has not yet performed the associated *getConfirm* to complete the passage.

In order to automatically transform the model into one in which such a measurement can be made we add a stochastic probe [7]. We specify the probe as:

$$Probe = browse, getConfirm \tag{1}$$

This is then automatically translated into the sequential PEPA component in Figure 2. The alphabet of the probe is the set of all activities observed by the probe component. The probe component is attached to the model via cooperation with some component of the model (perhaps the whole system). The cooperation set is the whole alphabet of the probe.

$$\begin{aligned}
Probe &\overset{def}{=} (browse, \top).Run + \underline{(getConfirm, \top).Probe} \\
Run &\overset{def}{=} (getConfirm, \top).Probe + \underline{(browse, \top).Run}
\end{aligned} \tag{2}$$

Fig. 2. The PEPA component automatically generated from the probe specification (1). The underlines are there only to indicate which prefix choices represent the *self-loops* and have no semantic meaning.

We do not wish to artificially block the observed component and so the probe component must be able to perform all activities in its alphabet in all states. Hence in each state of the generated probe a *self-loop* is added for any activity in the probe's alphabet which does not cause the probe to change its state. The self-loop observes the activity – allowing it to occur – but the probe does not alter its state. In the generated PEPA component in Figure 2 the self-loop activity observations have been underlined. In the remainder of this paper for the sake of space and clarity we will omit the self-loops.

The PEPA component automatically generated from the probe specification is then added to the model using the following transformation rule [8]:

$$\mathit{User}[?N] \Longrightarrow (\mathit{User} \underset{*}{\bowtie} \mathit{Probe})[?N]$$

Note that full cooperation '$*$' is used to ensure that the probe component observes all of its alphabet. By attaching a probe component to every user we can examine the population of users in the 'waiting' state. Here, the 'waiting' state is the state of having made at least one *browse* but not yet having performed the target action *getConfirm*. To do so we count the number of probes in the *Run* state. Because we have attached a probe to all the user components we term this a "*many probe*".

2.1 Partial Evaluation

Unfortunately, adding a probe component to every single user component in the model translates the original model from one which could be readily translated into a system of ordinary differential equations for analysis to one which cannot. This is because the translation into ODEs does not allow a cooperation to be replicated by the array operator. However, using an idea due to Nigel Thomas [25], we can use a technique called *partial evaluation* to turn the cooperation component $\mathit{User} \underset{*}{\bowtie} \mathit{Probe}$ into a sequential component $\mathit{User}_{\mathit{Probe}}$ which can then be used within the array operator as $\mathit{User}_{\mathit{Probe}}[N]$. The individual states of the $User$ and $Probe$ components are still recoverable from the states of the sequential $\mathit{User}_{\mathit{Probe}}$ component. This technique has been documented previously [6]. The $\mathit{User}_{\mathit{Probe}}$ component in our example is given by the PEPA code shown in Figure 3.

Having attached this *many probe* to the *User* component in our example model we can re-analyse the model and obtain the time series depicted in the graph in Figure 4. This gives us a steady-state of the system from which we can calculate the average response-time of our passage by:

$$\mathit{Pop}_{st}(\mathit{Run}) = \mathit{Pop}_{st}(\mathit{Browse}_{\mathit{Run}}) + \mathit{Pop}_{st}(\mathit{User}_{\mathit{Run}}) + \mathit{Pop}_{st}(\mathit{Buy}_{\mathit{Run}})$$

$$\begin{aligned} \mathit{AvgRes} &= \frac{\mathit{Pop}_{st}(\mathit{Run})}{\mathit{Thr}(\mathit{User}_{\mathit{Probe}}.\mathit{browse})} \\ &= \frac{\mathit{Pop}_{st}(\mathit{Run})}{\mathit{Pop}_{st}(\mathit{User}_{\mathit{Probe}}) \times \lambda_{br}} \end{aligned}$$

$$\begin{aligned}
User_{Probe} &\stackrel{\text{def}}{=} (browse, \lambda_{br}).Browse_{Run} + (buy, \lambda_{bu}).Buy_{Probe} \\
Buy_{Probe} &\stackrel{\text{def}}{=} (getConfirm, \lambda_{gc}).User_{Probe} \\
Browse_{Run} &\stackrel{\text{def}}{=} (getPage, \lambda_{gp}).User_{Run} \\
User_{Run} &\stackrel{\text{def}}{=} (browse, \lambda_{br}).Browse_{Run} + (buy, \lambda_{bu}).Buy_{Run} \\
Buy_{Run} &\stackrel{\text{def}}{=} (getConfirm, \lambda_{gc}).User_{Probe}
\end{aligned}$$

Fig. 3. The probed and partially evaluated *User* component

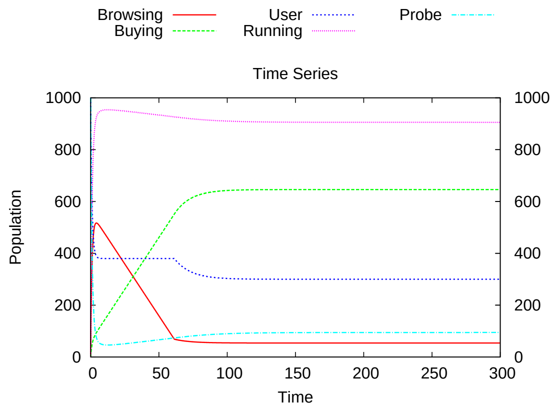

Fig. 4. The timeseries of the web model with a probe added to each user

In the next section we will consider how we can obtain a more detailed full response-time profile for the same passage.

3 Response-time Profiles

In this section we detail how to obtain a complete response-time profile from a PEPA model which we compile into a set of ODEs. We build upon the general idea originally proposed in [3]. This idea proposes that prior to generation of the ODEs all user components are transformed into components which enter a deadlocked state upon completion of the *response* activity that signals the end of the passage.

This transformation can be done by applying an *absorbing probe*. A regular probe definition such as that used in the previous section may be translated differently such that when a *response* activity is observed in the *Run* state (by which the end of the passage is signalled) the probe enters a deadlocked state rather than returning to the initial state of the probe. The probe definition for our example is translated into the absorbing probe:

$$Probe \stackrel{\text{def}}{=} (browse, \top).Run + (getConfirm, \top).Probe$$

$$\begin{aligned}
\mathit{Run} &\stackrel{\mathit{def}}{=} (\mathit{getConfirm}, \top).\mathit{Done} + (\mathit{browse}, \top).\mathit{Run} \\
\mathit{Done} &\stackrel{\mathit{def}}{=} \mathit{Stop}
\end{aligned}$$

In this translated probe component, when a probe component in the *Run* state performs a *getConfirm* it moves to the *Done* state. This represents the only change from the translated probe definition given in Figure 2. As before this probe definition is attached to each *User* component as $\mathit{User} \underset{*}{\bowtie} \mathit{Probe}$ which is then partially evaluated to give a $\mathit{User}_{\mathit{Probe}}$ component.

$$\begin{aligned}
\mathit{User}_{\mathit{Probe}} &\stackrel{\mathit{def}}{=} (\mathit{browse}, \lambda_{br}).\mathit{Browse}_{\mathit{Run}} + (\mathit{buy}, \lambda_{bu}).\mathit{Buy}_{\mathit{Probe}} \\
\mathit{Buy}_{\mathit{Probe}} &\stackrel{\mathit{def}}{=} (\mathit{getConfirm}, \lambda_{gc}).\mathit{User}_{\mathit{Probe}} \\
\mathit{Browse}_{\mathit{Run}} &\stackrel{\mathit{def}}{=} (\mathit{getPage}, \lambda_{gp}).\mathit{User}_{\mathit{Run}} \\
\mathit{User}_{\mathit{Run}} &\stackrel{\mathit{def}}{=} (\mathit{browse}, \lambda_{br}).\mathit{Browse}_{\mathit{Run}} + (\mathit{buy}, \lambda_{bu}).\mathit{Buy}_{\mathit{Run}} \\
\mathit{Buy}_{\mathit{Run}} &\stackrel{\mathit{def}}{=} (\mathit{getConfirm}, \lambda_{gc}).\mathit{User}_{\mathit{Done}} \\
\mathit{User}_{\mathit{Done}} &\stackrel{\mathit{def}}{=} \mathit{Stop}
\end{aligned}$$

Once the set of ODEs are generated, the populations of the user components are set such that all of them are in the source state of the passage. That is, the state which is entered when performing a *request* activity which signals the start of the passage. In this example the source state is the $\mathit{Browse}_{\mathit{Run}}$ state. We say that we *seed* the ODEs when we set the population of each component prior to the simulation of the set of ODEs. Seeding the ODEs will be used extensively in our approach.

Having added an absorbing probe and seeded the user component populations, an approximation to the Cumulative Distribution Function (CDF) of the passage specified by the probe specification can be obtained. The derived set of ODEs are evaluated by numerical integration giving the populations of each component type at time t, for each time value of the CDF that is required. The approximate value of the CDF at time t is given by:

$$CDF(t) = \frac{\mathit{User}_{\mathit{Done}}(t)}{\mathit{Browse}_{\mathit{Run}}(0)}$$

where $\mathit{User}_{\mathit{Done}}(t)$ is the number of probe components in the deadlocked absorbing state at time t and $\mathit{Browse}_{\mathit{Run}}(0)$ is the initial population of the source state. The initial population of the source state is equal to the original number of user components since we have set all of the user components into their respective source states.

In this method – which is a stochastic-probes specified version of the method proposed by Bradley *et al* – the state definitions $\mathit{User}_{\mathit{Probe}}$ and $\mathit{Buy}_{\mathit{Probe}}$ are unused because all of the user components are set to the source state and once the passage has begun it cannot be reset. We have identified two sources of inaccuracies. The first concerns the deadlocked user components. Such user components do not perform subsequent requests and responses and therefore do not compete with the still-to-complete probed components. Therefore the later completing components are afforded artificially exclusive access to shared resources. The second source of inaccuracy is that this method does not take into account when a request may occur. Because of this there may be initially too much competition for shared resources. In the following section we address both these issues.

3.1 One-Run Probes

We begin with our approach to solving the first source of inaccuracy by allowing user components which have completed their response-passage to continue to behave as a normal unprobed component. We must take care that we are able to distinguish those user components which have completed one passage whilst they are still able to perform their usual activities. We approach this task with the introduction of the notion of a *one-run* probe. Figure 5 shows a one-run probe together with a typical (cyclic) probe and the above described absorbing probe.

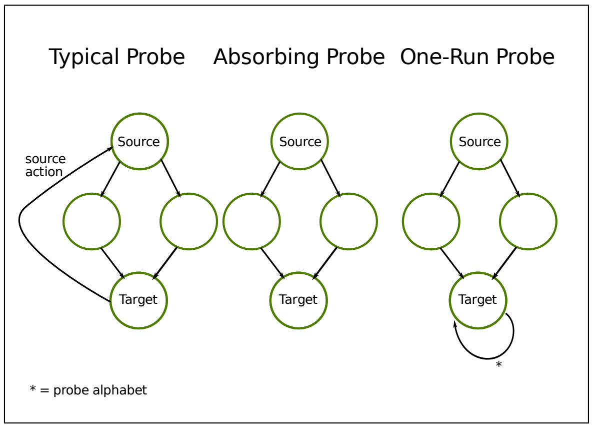

Fig. 5. A typical probe together with an absorbing probe and a one-run probe

As before with an absorbing probe we can use the same probe definition, but the translation into a PEPA component is modified such that upon entering the *Done* state the probe is still able to observe – and thus not block – all the activities within its own alphabet. The probe definition given above, and labelled (1), is translated into the following set of component definitions:

$$\begin{aligned} Probe &\stackrel{def}{=} (browse, \top).Run + (getConfirm, \top).Probe \\ Run &\stackrel{def}{=} (getConfirm, \top).Done + (browse, \top).Run \\ Done &\stackrel{def}{=} (browse, \top).Done + (getConfirm, \top).Done \end{aligned}$$

Once the one-run probe enters into its absorbing state – the *Done* state – it will never leave. Nonetheless, the component *Done* is able to passively cooperate over the activities in its alphabet, *browse* and *getConfirm*. Therefore each user component attached to a probe component is not blocked from behaving exactly as a non-probed user component. This has the consequence that the process *User* can perform the same activities as the process $(User \underset{*}{\bowtie} Done)$. Crucially, we are able to identify the population of user components which have completed the analysed passage using the population of probe components in the *Done* state. Thus we can tell how many user components have completed one response-passage even though they are now currently acting as a non-probed user component and offering up competition for the shared resources.

The diagram in Figure 6 depicts the general case of combining a one-run probe in cooperation with a component. In the bottom row the inner circles relate the state of the probe while the outer circles relate the state of the component. The area within the shaded rectangle represents a client component acting as though it were unprobed because the one-run probe is in the absorbing state.

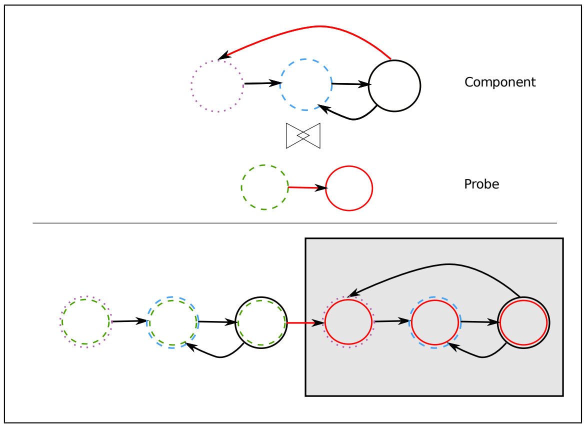

Fig. 6. A general case of combining a client component with a one-run probe component in cooperation.

Once we have added the one-run probe to our example model and partially evaluated the resulting cooperation between probe and user components we obtain the following PEPA model:

$$
\begin{aligned}
Browse_{Run} &\overset{def}{=} (getPage, \lambda_{gp}).User_{Run} \\
User_{Run} &\overset{def}{=} (browse, \lambda_{br}).Browse_{Run} + (buy, \lambda_{bu}).Buy_{Run} \\
Buy_{Run} &\overset{def}{=} (getConfirm, \lambda_{gc}).User_{Ran}
\end{aligned}
$$

$$\begin{aligned}
\mathit{User}_{\mathit{Ran}} &\stackrel{\mathit{def}}{=} (\mathit{browse}, \lambda_{br}).\mathit{Browse}_{\mathit{Ran}} + (\mathit{buy}, \lambda_{bu}).\mathit{Buy}_{\mathit{Ran}} \\
\mathit{Browse}_{\mathit{Ran}} &\stackrel{\mathit{def}}{=} (\mathit{getPage}, \lambda_{gp}).\mathit{User}_{\mathit{Ran}} \\
\mathit{Buy}_{\mathit{Ran}} &\stackrel{\mathit{def}}{=} (\mathit{getConfirm}, \lambda_{gc}).\mathit{User}_{\mathit{Ran}}
\end{aligned}$$

$$\mathit{Server} \stackrel{\mathit{def}}{=} (\mathit{getPage}, \lambda_{sp}).\mathit{Server} + (\mathit{getConfirm}, \lambda_{sc}).\mathit{Server}$$

$$\begin{aligned}
\mathit{System} &\stackrel{\mathit{def}}{=} (\mathit{Server}[M]) \underset{\mathcal{L}}{\bowtie} \mathit{Browse}_{\mathit{Run}}[N] \\
&\qquad \text{where } \mathcal{L} = \{\, \mathit{getPage}, \mathit{getConfirm} \,\}
\end{aligned}$$

The definitions $\mathit{Browse}_{\mathit{Run}}$, $\mathit{User}_{\mathit{Run}}$ and $\mathit{Buy}_{\mathit{Run}}$ correspond to a user component which is attached to a probe component in the *Run* state. The definitions $\mathit{Browse}_{\mathit{Ran}}$, $\mathit{User}_{\mathit{Ran}}$ and $\mathit{Buy}_{\mathit{Ran}}$ correspond to a user component in cooperation with a probe component in the *Done* state. We can therefore evaluate the number of user components which have completed the passage using:

$$\mathit{Done}(t) = \mathit{User}_{\mathit{Ran}}(t) + \mathit{Browse}_{\mathit{Ran}}(t) + \mathit{Buy}_{\mathit{Ran}}(t)$$

We may therefore approximate the CDF at time t with:

$$\mathit{CDF}(t) = \frac{\mathit{Done}(t)}{\mathit{Browse}_{\mathit{Run}}(0)} \tag{3}$$

However, this would calculate a pessimistic CDF due to the second problem identified above – the state of the model at the time a request is made is not utilised with the method thus far stated. The solution is to use the results of steady-state analysis on the original model to seed the ODEs derived from the probed model. We performed steady-state analysis using the ODEs derived from the original model. From this we obtained the values for the populations of the user states shown in the left-hand columns of the table in Figure 7.

User	300
Browse	54
Buy	646

$\mathit{User}_{\mathit{Run}}$	0
$\mathit{Browse}_{\mathit{Run}}$	54
$\mathit{Buy}_{\mathit{Run}}$	0
$\mathit{User}_{\mathit{Ran}}$	300
$\mathit{Browse}_{\mathit{Ran}}$	0
$\mathit{Buy}_{\mathit{Ran}}$	646

Fig. 7. The left-hand columns show the steady-state populations of the *User* component states. The right-hand columns show how the derived ODEs are seeded in order to obtain a CDF for the response-passage.

The central idea is that when we seed the derived ODEs the behaviour of the system should be that of the steady-state. In particular the source state of the response passage has only the number of components expected to be in that state at any given time. This is in contrast to before when the entire population of user components was set to be in the source state of the response passage. The remaining population of user components is distributed about the user states as in the steady-state. However, this remaining population must not interfere with the calculation of the CDF. In particular we must not count any passages subsequently started by this user population. The solution is that these uncounted users are seeded in cooperation with a completed probe component. This means that we have a number of already completed passages before the simulation of the seeded ODEs begins. We know the number of such passages and can therefore subtract them from the end count.

The columns on the right half of the table in Figure 7 show how we seed the probed model in order to approximate the CDF. All those user states which are not the source state have their populations given to their respective states in a user component in cooperation with a probe which is in the *Done* state. The source state of the original unprobed model *Browse* has all of its steady-state population given to the equivalent *Browse* state in which the probe is in the *Run* state, i.e. the $\mathit{Browse}_{\mathit{Run}}$ state. To calculate the number of user components which have completed a passage we subtract the initial number of completed probes from the number of completed probes at time t:

$$\begin{aligned}\mathit{Done}(t) = {} & (\mathit{User}_{\mathit{Ran}}(t) + \mathit{Browse}_{\mathit{Ran}}(t) + \mathit{Buy}_{\mathit{Ran}}(t)) \\ & - (\mathit{User}_{\mathit{Ran}}(0) + \mathit{Buy}_{\mathit{Ran}}(0) + \mathit{Browse}_{\mathit{Ran}}(0))\end{aligned}$$

Note that $\mathit{Browse}_{\mathit{Ran}}(0)$ will of course be zero since this corresponds to the source state of the user component. The approximated CDF is then calculated as before with equation (3).

3.2 Validation

In order to validate our computed CDF we can attempt to solve the model using translation into a CTMC. Unfortunately the reason for translating the model into a system of ODEs in the first place was due to the fact that the model was too large for analysis via CTMC. Previously whenever a model was too large for analysis via CTMC, but a response-time profile rather than an average response-time was required we resorted to abstracting the model. A common method of abstraction is for the behaviour of only one single client component to be fully modelled while the behaviour of all other clients is aggregated into one super client (in some cases a few super clients). Figure 8 shows a model which is an approximation of the original model but which has a state-space small enough to allow analysis via CTMC. The main trick is to reduce the rate at which the server may respond by dividing the rate by the steady-state population of user clients waiting for such a response. The steady-state population is obtained by evaluating the original model using ODEs.

However this means that we are comparing one approximation to another. The first approximation is an approximation to the response-profile analysis of the full model whilst the second is a full response-profile for a model which is an

$$\begin{aligned}
\lambda_{gp} &= \lambda_{sp}/\mathit{Browse}(\infty) \\
\lambda_{gc} &= \lambda_{sc}/\mathit{Buy}(\infty)
\end{aligned}$$

$$\begin{aligned}
\mathit{User} &\overset{\mathit{def}}{=} (\mathit{browse}, \lambda_{br}).\mathit{Browse} + (\mathit{buy}, \lambda_{bu}).\mathit{Buy} \\
\mathit{Browse} &\overset{\mathit{def}}{=} (\mathit{getPage}, \lambda_{gp}).\mathit{User} \\
\mathit{Buy} &\overset{\mathit{def}}{=} (\mathit{getConfirm}, \lambda_{gc}).\mathit{User}
\end{aligned}$$

$$\mathit{Server} \overset{\mathit{def}}{=} (\mathit{getPage}, \lambda_{sp}).\mathit{Server} + (\mathit{getConfirm}, \lambda_{sc}).\mathit{Server}$$

$$\begin{aligned}
\mathit{System} &\overset{\mathit{def}}{=} (\mathit{Server}[M]) \underset{\mathcal{L}}{\bowtie} (\mathit{User}) \\
&\quad \text{where } \mathcal{L} = \{\, \mathit{getPage}, \mathit{getConfirm} \,\}
\end{aligned}$$

Fig. 8. An approximation model solvable via a CTMC

approximation to the full model. To bridge this gap we use stochastic simulation. The simulation is performed a number of times giving a percentage of runs in which a target state is entered before any given time t up to some maximum. Obviously the greater the number of runs the more accurate the computed CDF is, but the longer the analysis takes.

A stochastic simulator can only be used for a response-profile if there are a fixed number of source states and we know the relative probabilities that the passage is begun in each of the given source states. When analysing via CTMC analysis this information is achieved by exploring the whole state-space of the system and then solving the embedded Markov chain of the entire state-space. Since we wish to use this technique on a model with an infeasibly large state-space this approach is not open to us. An alternative is to use a single source state in the belief that it is representative of all source states. Some of the alternative source states will give rise to a more optimistic response-profile and others a more pessimistic one. We require that these two possibilities cancel each other out when scaled by their respective probabilities of being the particular source state in which a particular response passage is begun.

We must single out an individual client for this style of analysis so the PEPA model which we in turn automatically translated into a Java simulator is given in Figure 9. This provides a single tagged client and it is the response time as observed by that tagged client that we measure.

This provides a specific source state and a specific condition for the target states. The source state as mentioned above is to have the steady-state populations for the non-tagged clients and the tagged client in its source state: Browse_{tag}. The seeded population of the untagged client state *Browse* is reduced by one as this corresponds to the single tagged client. The target condition is simple; the tagged client must be in its target state, Done_{tag}. In this example

$$
\begin{aligned}
User_{tag} &\stackrel{\mathit{def}}{=} (browse, \lambda_{br}).Browse_{tag} + (buy, \lambda_{bu}).Buy_{tag} \\
Browse_{tag} &\stackrel{\mathit{def}}{=} (getPage, \lambda_{gp}).User_{tag} \\
Buy_{tag} &\stackrel{\mathit{def}}{=} (getConfirm, \lambda_{gc}).Done_{tag} \\
Done_{tag} &\stackrel{\mathit{def}}{=} Stop
\end{aligned}
$$

$$
\begin{aligned}
User &\stackrel{\mathit{def}}{=} (browse, \lambda_{br}).Browse + (buy, \lambda_{bu}).Buy \\
Browse &\stackrel{\mathit{def}}{=} (getPage, \lambda_{gp}).User \\
Buy &\stackrel{\mathit{def}}{=} (getConfirm, \lambda_{gc}).User
\end{aligned}
$$

$$
Server \stackrel{\mathit{def}}{=} (getPage, \lambda_{sp}).Server + (getConfirm, \lambda_{sc}).Server
$$

$$
\begin{aligned}
System &\stackrel{\mathit{def}}{=} (Server[M]) \underset{\mathcal{L}}{\bowtie} (User[N-1] \parallel User_{tag}) \\
&\quad \text{where } \mathcal{L} = \{\, getPage, getConfirm \,\}
\end{aligned}
$$

Fig. 9. A model suitable for generating a custom Java simulator using which a response-time profile can be obtained.

there is exactly one source state; in Section 4 we analyse a passage with multiple source states.

There are now three results to compare, the first is our approximate CDF calculated using ODEs derived from the original model. The second is a precise CDF calculated from the CTMC derived from an approximation to the original model. The third is a CDF obtained through stochastic simulation of the original model. The results of the comparison are shown in the graph in Figure 10. There is very little disagreement between all three methods. To calculate these results – on an typical desktop machine – our ODE approach takes less than one second. For the simulation technique we set the number of independent runs to ten thousand and this took 123 seconds. The analysis of the CTMC took 1,065 seconds. This is because the approximation model gives rise to a CTMC with large rates and the uniformisation technique must perform many matrix multiplications in the presence of large rates.

4 Multiple Source States

So far the passages we have been concerned with have all had the distinguishing feature of a single source state meaning that at the beginning of a response-passage the user/client component state is known. Where this has been the case, we have been able to use the steady-state distribution to define the initial value problem for the derived set of ODEs. This approach is not applicable if there are multiple source states within the client component. Multiple source states occur if the user component may perform an activity which begins the

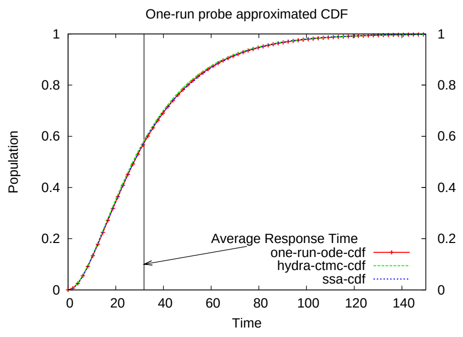

Fig. 10. Comparison of the CDF computed via the approximation model and CTMC analysis and the CDF computed via the original model and ODE analysis.

response-passage and transition into one of a choice of two or more states. This means there must be more than one *request* activity – although these may have the same name. For example, if the activity *request* begins a response-passage then the following component has two source states, namely, $Wait_1$ and $Wait_2$:

$$\begin{aligned} Client &\overset{def}{=} (request, \lambda_1).Wait_1 + (request, \lambda_2).Wait_2 \\ Wait_1 &\overset{def}{=} (response, \delta_1).Client \\ Wait_2 &\overset{def}{=} (response, \delta_2).Client \end{aligned} \tag{4}$$

Recall our original model from Section 2 shown in Figure 1. Suppose now we wish to measure the general responsiveness of the system, rather than that of either the *browse* or the *buy* activities in isolation. To do this we may analyse the response passage from an occurrence of either kind of request to the occurrence of its associated response. This can be specified with the probe specification:

$$Probe = (browse, getPage) \mid (buy, getConfirm) \tag{5}$$

This is then automatically translated into the following one-run probe component definition:

$$\begin{aligned} Probe &\overset{def}{=} (browse, \top).Run_1 + (buy, \top).Run_2 \\ Run_1 &\overset{def}{=} (getPage, \top).Done_1 \\ Run_2 &\overset{def}{=} (getConfirm, \top).Done_2 \\ Done_1 &\overset{def}{=} (browse, \top).Done_1 + (buy, \top).Done_1 \\ &\quad + (getPage, \top).Done_1 + (getConfirm, \top).Done_1 \\ Done_2 &\overset{def}{=} \text{as } Done_1 \end{aligned} \tag{6}$$

When this probe definition is attached to a user component, it creates a component which always behaves as a usual unprobed user component. However when

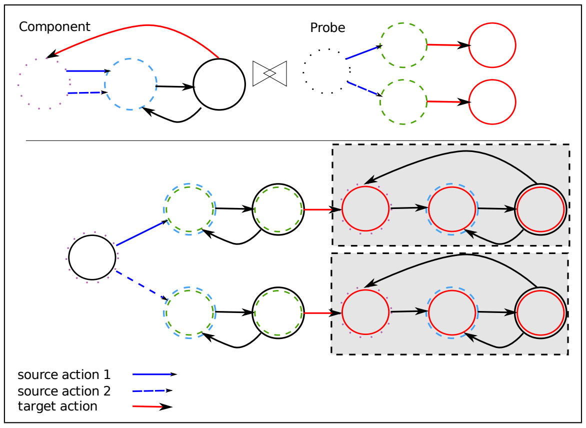

Fig. 11. The general case for combining a component with a one-run probe for the case in which the response-passage has multiple user component source states

the first request/source activity is performed by the user component – and hence observed by the probe component – then the state of the probe component allows us to determine which activity began the passage and whether or not the response-passage has been completed. The general case for attaching a one-run probe with multiple source states to a user component is depicted in Figure 11.

When the probe definition (6) is added to the model we obtain the following PEPA model where the X may be replaced by both 1 and 2.

$$\begin{aligned}
\mathit{UserProbe} &\overset{\mathit{def}}{=} (\mathit{browse}, \lambda_{br}).\mathit{Browse}_{\mathit{Run}1} + (\mathit{buy}, \lambda_{bu}).\mathit{Buy}_{\mathit{Run}2} \\
\mathit{Browse}_{\mathit{Run}1} &\overset{\mathit{def}}{=} (\mathit{getPage}, \lambda_{gp}).\mathit{User}_{\mathit{Ran}1} \\
\mathit{Buy}_{\mathit{Run}2} &\overset{\mathit{def}}{=} (\mathit{getConfirm}, \lambda_{gc}).\mathit{User}_{\mathit{Ran}2}
\end{aligned}$$

$$\begin{aligned}
\mathit{User}_{\mathit{Ran}X} &\overset{\mathit{def}}{=} (\mathit{browse}, \lambda_{br}).\mathit{Browse}_{\mathit{Ran}X} + (\mathit{buy}, \lambda_{bu}).\mathit{Buy}_{\mathit{Ran}X} \\
\mathit{Browse}_{\mathit{Ran}X} &\overset{\mathit{def}}{=} (\mathit{getPage}, \lambda_{gp}).\mathit{User}_{\mathit{Ran}X} \\
\mathit{Buy}_{\mathit{Ran}X} &\overset{\mathit{def}}{=} (\mathit{getConfirm}, \lambda_{gc}).\mathit{User}_{\mathit{Ran}X}
\end{aligned}$$

$$\mathit{Server} \overset{\mathit{def}}{=} (\mathit{getPage}, \lambda_{sp}).\mathit{Server} + (\mathit{getConfirm}, \lambda_{sc}).\mathit{Server}$$

$$\mathit{System} \overset{\mathit{def}}{=} (\mathit{Server}[M]) \underset{\mathcal{L}}{\bowtie} (\mathit{Browse}_{\mathit{Run}}[N])$$

$$\text{where } \mathcal{L} = \{\, \mathit{getPage}, \mathit{getConfirm} \,\}$$

After the addition of the probe, the model is translated into a set of ODEs which are then evaluated against the initial value problem using numerical integration. As before, we must seed the ODEs by setting the correct populations of component states to represent the steady-state of the system. In particular we set the number of clients in each of the source states of the system depending on the steady-state, not how often each kind of request occurs. Recall that we already know the steady-state solution of this model from the original model 7 since the addition of the one-run probes does not change the behaviour of the model. We use the same idea by taking the source populations from the steady-state. Essentially the seeding is the same but now the *Buy* user state is also a source state. Therefore rather than the steady-state population of the *Buy* state being seeded as Buy_{Ran} it is seeded as Buy_{Run2}. Finally the steady-state population of the *User* state must be initialised as either $User_{Ran1}$ or $User_{Ran2}$. The purpose of this, as before, is that we have a user population which can be excluded from the CDF approximation calculation but which still contributes to the competition for the shared resources. It does not matter which of the two as we can subtract the sum $User_{Ran1}(0) + User_{Ran2}(0)$ for which we can use the shorthand $User_{RanX}(0)$.

The naïve way to compute the CDF would be to take the value at time t equal to the number of probes completed divided by the number of probes in total which is equal to those which started in the two source states, or:

$$CDF(t) = \frac{User_{RanX}(t) + Browse_{RanX}(t) + Buy_{RanX}(t) - User_{RanX}(0)}{Browse_{Run1}(0) + Buy_{Run1}(0)}$$

This would give a CDF value which over-compensated for the effects of the *buy* response-passages. Because although ‘buy’ response passages are started much less frequently then ‘browse’ response passages there are more of them in the calculation. This is because the user client spends longer in the source state of a ‘buy’ passage than in the source state of a ‘browse’ passage. However, each probe’s absorbing state depends on how the passage was started. This allows us to scale the contribution of each kind of passage according to how likely it is to occur. We record the throughput of the *User.buy* and *User.browse* actions and hence their ratios:

$$Ratio\_browse = \frac{Thr(User.browse)}{Thr(User.buy) + Thr(User.browse)}$$

That is, the fraction of requests which are browse requests. In other words, the probability that a general response-passage is begun by entering the *Browse* state. This allows us to independently calculate the probability of completing each kind of request and then scale the probabilities according to their frequencies. We have that the probability of completing a *browse* response-passage at time t is equal to:

$$CDF_{bro}(t) = \frac{User_{Ran1}(1) + Browse_{Ran1}(t) + Buy_{Ran1}(t) - User_{Ran1}(0)}{Browse_{Run1}(0)}$$

Similarly for the *buy* response-passages:

$$CDF_{buy}(t) = \frac{User_{Ran2}(1) + Browse_{Ran2}(t) + Buy_{Ran2}(t) - User_{Ran2}(0)}{Buy_{Run2}(0)}$$

Finally, our CDF for general requests in our model is calculated by:

$$CDF(t) = Ratio\_browse \times CDF_{bro}(t) + Ratio\_buy \times CDF_{buy}(t)$$

4.1 Validation

For comparison the probe specification (5) can be translated in the traditional cyclic manner and used to analyse the same passage via translation to a CTMC. As before, the full model cannot be translated to a CTMC because of the state-space size but may be approximated by the model in Figure 8. Plotted in Figure 12 are the results from CTMC analysis of the approximation model. The plotted lines are the CDF functions of three response-passages, that of all *browse* requests, that of all *buy* requests and the passage of current interest i.e. all requests in general. These clearly show the profile of the general response is a combination of the two response-passages of which it consists. The *browse* requests are responded to very quickly with high probability corresponding to the steepest line of the graph. In contrast, the probability of completing a *buy* response-passage increases much less sharply. The general response then behaves in two modes initially a large proportion of the general requests (all the *browse* requests) are responded to very quickly with high probability. But then once we reach the probability corresponding to the proportion of general requests which are *browse* requests the gradient of the CDF becomes similar to the gradient of the *buy* requests. This is therefore a useful CDF to have computed showing that general requests do indeed behave in two separate modes. For the simple model we have analysed here it was easy to predict this but in general computing such a CDF can be informative.

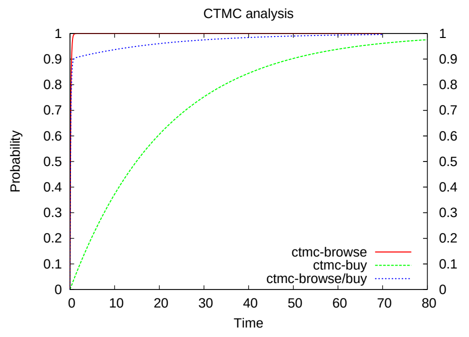

Fig. 12. The CDFs for the web model as calculated by the analysis of the CTMC derived from the approximation model

The top graph in Figure 13 shows how well our method of response-profile analysis using ODEs for this particular query does. We see that for general requests we do indeed compute a CDF showing two distinct phases or modes corresponding to the contrasting behaviours of the two kinds of response-passages of which the general response-passage consists. We see that it compares quite reasonably with that of the CDF computed via translation of the approximation model to a CTMC. Recall that this is only an approximation itself and therefore exact agreement is not to be expected or even desired. Also shown in this graph is the ODE-computed CDF for just the *buy* response passages and the CDF computed via translation to ODE in the naïve manner of the previous section which does not adjust for multiple source states. We see that these two are similar because the naïve method vastly over-attributes the contribution of *buy* response-passages to the overall general response-passage profile. Finally we have again used our technique of generating a stochastic simulator to evaluate the response-passage on the full sized model. The CDF calculated via this method is also shown on this graph and in the bottom graph of Figure 13. We have zoomed in on the first five time units to show how the three methods of response-time calculation compare. In this we see that the two methods analysing the full-sized model agree (slightly) more than the CTMC method analysing an approximated model.

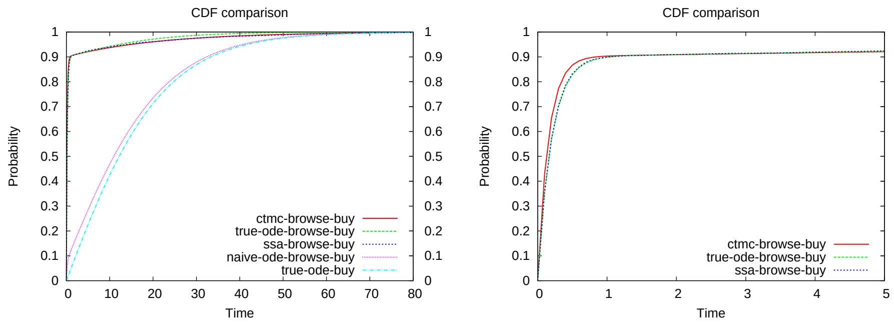

Fig. 13. Comparison of the approximated CDFs for the response passage begun with the browse and buy activities. Our 'true' version approximates well that of the CTMC whereas the naïve version is too pessimistic. We plot also the CDF approximation using our technique for the response passage begun with only the 'buy' activity. This is similar to the naïve version suggesting that the naïve version exaggerates the contribution of the 'buy' response passages to the overall CDF.

5 Conclusions

In this paper we have investigated the use of the PEPA language to obtain an approximation to the response-time profiles of passages for models with very large state-spaces. Traditionally PEPA models have been analysed by translation into the underlying CTMC. Recently, to avoid the state-space explosion problem, modellers have taken advantage of an automatic translation from PEPA models into a set of ODEs which approximate the CTMC underlying the PEPA model. The obtained ODEs have then been evaluated via numerical integration to obtain a time series plotting the population of each component type against time. Evaluation is continued until the populations of each component kind are stable (the stationary points of the differential equation). These stable populations have then been taken to be an approximation to the steady-state of the model. This then gives a way to infer the average response-time of modelled systems and services for large-scale systems.

For many models a more detailed analysis is required, and in the CTMC world transient analysis of the CTMC has allowed for the calculation of response-time quantiles which plot the probability of completion against time. This paper has described a technique to allow an approximation to this response-time profile to be calculated from the ODEs derived from a PEPA model, thus allowing response-time profiles to be approximated for models with a state-space size that is far beyond the feasibility for CTMC analysis.

The use of the stochastic probe query specification language is a key element in bringing the technology of our method to the modeller. In particular it is possible to use the same query specification to analyse the model regardless of whether it is then translated to a CTMC or a set of ODEs. We believe our method is robust and that the subset of passage-time queries for which it is appropriate — those we have termed response-passages — is large enough to be applicable in many situations. We know that some passages which may not appear to fall within this framework can be altered via the addition of observation probes in order to become response-passages although this is a topic for future consideration.

Although the work described here decreases the gap between those analyses available only via translation to a CTMC and those available via ODE analysis there are still some queries (not just general passage-time queries but other work done on CTMCs) which can only be performed via translation to a CTMC. This is sometimes done by first constructing a model of the real system and then approximating that with a model with a smaller (tractable) state-space by abstracting some of the behaviour of the original model. This was done in our two examples in Section 3. A further benefit of our work here is that such an approximation model may be validated by computing response-time profiles for both the original model using our method and for the approximation model using transient analysis on the derived CTMC.

We note that the method considered here is not universally applicable. The main deficiency is that analysis via a CTMC allows each individual source state to contribute to the computed CDF. Since we cannot enumerate all of the source states our method implicitly approximates this by using only one source state which corresponds to the steady-state of the system. In the context of a request-response passage this means that we assume that all requests are made when the system is under steady-state conditions. Requests made outside these conditions are either very unlikely or such requests cancel each other out. That is, a request made when the system is under low usage has a high probability of completing early but conversely a request made when the system is under high usage has a low probability of completing early. Given these constraints, it is relatively simple to come up with a model which will induce our system to approximate response-time profiles badly. Constructing such a system is particularly trivial if functional rates are allowed within the model. However we argue that in this case the modeller knows that they are deliberately changing the responsiveness of the system under differing loads. We therefore believe that the modeller may analyse the response-times in those conditions separately and, if necessary, combine the results.

We have only considered translating the model into a system of ODEs, but for large models another method to avoid the generation of a large state-space is to use simulation. This has been used before for PEPA and since it is possible to obtain a time series analysis using simulation we see no reason why the same technique as described here cannot be used to obtain response-time profiles. Additionally, passage-time analysis was extended for PEPA to include analysis of the end of the passage in question resulting in passage-end [5] analysis. Because of the machinery required here to allow for multiple source states, which is similar to that required for passage-end analysis, we believe our method extends nicely to allow passage-end analysis to be performed over models with massive state-spaces. Such analysis can be important in the analysis of service level-agreements. Passage-time analysis for service level-agreements (SLAs) is important as it allows such statements as: "Ninety percent of requests are responded to within 10 seconds". However in the past we found that such analysis meant that a simple way to achieve an SLA would be to respond negatively (eg "request rejected" or "service too busy") to all requests. Passage-end analysis allows more succinct SLAs to be evaluated such as: "Ninety percent of *positive* requests are responded to within 10 seconds" or "Ninety percent of requests are responded to within 10 seconds and of those at least eighty percent are positive".

6 Acknowledgements

The authors have been supported by the EU FET-IST Global Computing 2 project SENSORIA (Software Engineering for Service-Oriented Overlay Computers (IST-3-016004-IP-09)). The authors would like to thank Professor Jane Hillston for constructive feedback.

References

1. Argent-Katwala, A., Clark, A., Foster, H., Gilmore, S., Mayer, P., Tribastone, M.: Safety and response-time analysis of an automotive accident assistance service. In: Proceedings of the 3rd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2008). pp. 191–205. No. 17 in Communications in Computer and Information Science (CCIS), Springer-Verlag, Porto Sani, Greece (Oct 2008)

2. Bradley, J., Gilmore, S.: Stochastic simulation methods applied to a secure electronic voting model. Electr. Notes Theor. Comput. Sci. 151(3), 5–25 (2006)
3. Bradley, J.T., Hayden, R., Knottenbelt, W.J., Suto, T.: Extracting Fluid Response times from PEPA models. In: PASTA'08, 7th Workshop on Process Algebra and Stochastically Timed Activities (July 2008), http://pubs.doc.ic.ac.uk/responsetimes-fluidpepa/, this is a short version summary of the full paper that appeared at SIPEW 2008 (http://pubs.doc.ic.ac.uk/responsetimes-fluidanalysis).
4. Clark, A., Gilmore, S.: Terminating passage-time calculations on uniformised Markov chains. In: Argent-Katwala, A., Dingle, N.J., Harder, U. (eds.) Proceedings of the Twenty-Fourth annual UK Performance Engineering Workshop. pp. 64–75 (Jun 2008)
5. Clark, A., Duguid, A., Gilmore, S.: Passage-end analysis. In: Bradley, J.T. (ed.) Computer Performance Engineering, 6th European Performance Engineering Workshop, EPEW 2009, London, UK, July 9-10, 2009, Proceedings. Lecture Notes in Computer Science, vol. 5652, pp. 110–115. Springer (2009)
6. Clark, A., Duguid, A., Gilmore, S., Tribastone, M.: Partial evaluation of PEPA models for fluid-flow analysis. In: Thomas, N., Juiz, C. (eds.) Proceedings of the 5th European Performance Engineering Workshop (EPEW 2008). LNCS, vol. 5261, pp. 2–16. Springer, Palma de Mallorca, Spain (Sep 2008)
7. Clark, A., Gilmore, S.: State-aware performance analysis with eXtended Stochastic Probes. In: Thomas, N., Juiz, C. (eds.) Proceedings of the 5th European Performance Engineering Workshop (EPEW 2008). LNCS, vol. 5261, pp. 125–140. Springer, Palma de Mallorca, Spain (Sep 2008)
8. Clark, A., Gilmore, S.: Transformations in PEPA Models and Stochastic Probe Placement. In: Djemame, K. (ed.) Proceedings of the Twenty-Fifth UK Performance Engineering Workshop. pp. 1–16. Leeds University (Jul 2009)
9. Dingle, N.J.: Parallel Computation of Response Time Densities and Quantiles in Large Markov and Semi-Markov Models. Ph.D. thesis, Department of Computing, Imperial College London. University of London. (October 2004), http://pubs.doc.ic.ac.uk/njdthesis/
10. Gillespie, D.: Exact stochastic simulation of coupled chemical reactions. Journal of Physical Chemistry 81(25), 2340–2361 (December 1977)
11. Gilmore, S., Hillston, J., Ribaudo, M.: An efficient algorithm for aggregating PEPA models. IEEE Transactions on Software Engineering 27(5), 449–464 (May 2001)
12. Grassmann, W.: Transient solutions in Markovian queueing systems. Computers and Operations Research 4, 47–53 (1977)
13. Gross, D., Miller, D.: The randomization technique as a modelling tool and solution procedure for transient Markov processes. Operations Research 32, 343–361 (1984)
14. Harrison, P.G., Knottenbelt, W.J.: Passage time distributions in large markov chains. In: SIGMETRICS '02: Proceedings of the 2002 ACM SIGMETRICS international conference on Measurement and modeling of computer systems. pp. 77–85. ACM, New York, NY, USA (2002)
15. Hillston, J.: The nature of synchronisation. In: Herzog, U., Rettelbach, M. (eds.) Proceedings of the Second International Workshop on Process Algebras and Performance Modelling. pp. 51–70. Erlangen (Nov 1994)
16. Hillston, J.: A Compositional Approach to Performance Modelling. Cambridge University Press (1996)
17. Hillston, J.: Fluid flow approximation of PEPA models. In: Proceedings of the Second International Conference on the Quantitative Evaluation of Systems. pp. 33–43. IEEE Computer Society Press, Torino, Italy (Sep 2005)

18. Hillston, J.: Process algebras for quantitative analysis. In: Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science (LICS' 05). pp. 239–248. IEEE Computer Society Press, Chicago (Jun 2005)
19. Hillston, J.: Tuning systems: From composition to performance. The Computer Journal 48(4), 385–400 (May 2005), the Needham Lecture paper
20. Hillston, J., Kloul, L.: An efficient Kronecker representation for PEPA models. In: de Alfaro, L., Gilmore, S. (eds.) Proceedings of the first joint PAPM-PROBMIV Workshop. Lecture Notes in Computer Science, vol. 2165, pp. 120–135. Springer-Verlag, Aachen, Germany (Sep 2001)
21. Katoen, J.P., Zapreev, I.S.: Simulation-Based CTMC Model Checking: An Empirical Evaluation. In: Quantitative Evaluation of Systems (QEST). pp. 31–40. IEEE Computer Society (2009), www.mrmc-tool.org
22. Little, J.D.C.: A proof of the queueing formula $l = \lambda w$. Operations Research 9, 380–387 (1961)
23. Melamed, B., Yadin, M.: Randomization procedures in the computation of cumulative-time distributions over discrete state markov processes. Operations Research 32(4), 926–944 (1984), http://www.jstor.org/stable/170587
24. Ribaudo, M.: On the aggregation techniques in stochastic Petri nets and stochastic process algebras. In: Gilmore, S., Hillston, J. (eds.) Proceedings of the Third International Workshop on Process Algebras and Performance Modelling. pp. 600–611. Special Issue of *The Computer Journal*, 38(7) (Dec 1995)
25. Zhao, Y., Thomas, N.: Approximate solution of a PEPA model of a key distribution centre. In: Kounev, S., Gorton, I., Sachs, K. (eds.) Performance Evaluation: Metrics, Models and Benchmarks, SPEC International Performance Evaluation Workshop, SIPEW 2008, Darmstadt, Germany, June 27-28, 2008. Proceedings. Lecture Notes in Computer Science, vol. 5119, pp. 44–57. Springer (2008)