Stochastic Process Algebras: From Individuals to Populations

Jane Hillston[1], Mirco Tribastone[2] and Stephen Gilmore[1]

[1] *School of Informatics, University of Edinburgh, Edinburgh, UK*
[2] *Institut für Informatik, Ludwig-Maximilians-Universität, Munich, Germany*
Email: jeh@inf.ed.ac.uk, tribastone@pst.ifi.lmu.de, stg@inf.ed.ac.uk

In this paper we report on progress in the use of stochastic process algebras for representing systems which contain many replications of components such as clients, servers and devices. Such systems have traditionally been difficult to analyse even when using high-level models because of the need to represent the vast range of their potential behaviour. Models of concurrent systems with many components very quickly exceed the storage capacity of computing devices even when efficient data structures are used to minimise the cost of representing each state. Here, we show how population-based models which make use of a continuous approximation of the discrete behaviour can be used to efficiently analyse the temporal behaviour of very large systems via their collective dynamics. This approach enables modellers to study problems which cannot be tackled with traditional discrete-state techniques such as continuous-time Markov chains.

Keywords: Performance modelling, stochastic process algebra, PEPA, compositional modelling, continuous approximation

Received 29th March 2011; revised 04th July 2011; accepted 20th August 2011

1. INTRODUCTION

Stochastic process algebras have been successfully used to construct performance models of a wide variety of systems for more than a decade [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]. The strength of these modelling languages stems from their ability to express a high-level compositional description of a system from which an underlying mathematical model can be rigorously derived. The mathematical model can then be subjected to analysis to reveal performance properties of the system, readily presented to the modeller in terms of the original high-level compositional description.

This modelling approach was first defined using an underlying mathematical model which is a Continuous-Time Markov Chain (CTMC), which was subjected to numerical solution in order to derive performance indices. The advantage of this was that the stochastic process algebra was able to take advantage of a rich corpus of results and algorithms relating to the solution of CTMCs. The disadvantage was that the CTMC, like all discrete state space representations, is prone to problems of *state space explosion*. Thus while it is well-understood how to solve arbitrary CTMCs from a theoretical point of view, from a pragmatic perspective there exists a size of state space beyond which solution becomes infeasible [11].

In recent years we have been exploring ways to give alternative mathematical interpretations to the high-level system descriptions produced by one stochastic process algebra, PEPA. This alternative interpretation can be viewed as forming an approximation of the CTMC generated by the usual interpretation or as taking a more abstract interpretation of the original PEPA model. These two views have been proven to coincide. The type of models which are suitable for this alternative treatment are those in which we have large numbers of repeated components. Thus it is particularly appropriate for studying the scalability of software systems under increasing load. This is a common concern, particularly within service-oriented computing. In this context the behaviour of the system as a whole can be regarded as an instance of *collective dynamics*, i.e. the behaviour at the population level results from interactions which take place between individuals within the population.

The remainder of the paper is structured as follows. In Section 2 we introduce stochastic process algebras in more detail, focussing particularly on PEPA in Subsection 2.1. The interpretation of a PEPA model as an underlying mathematical model is considered in Section 3, where we explain how different levels of abstraction can lead to different mathematical interpretations, and the relationships between these alternative mathematical models. The approach is illustrated by a large example, taken from the domain of service-oriented computing, in Section 4. Section 5 presents model evaluation with a particular focus on use

of the PEPA Eclipse Plug-in. Related work is presented in Section 6. Finally, in Section 7 we discuss some areas for further work.

2. STOCHASTIC PROCESS ALGEBRAS

Stochastic process algebras emerged in the early 1990s as a compositional modelling formalism for constructing models suitable for performance evaluation based on CTMCs. Queueing networks had been extensively used for performance modelling in the preceding two decades but the structures and complexities of distributed systems presented difficulties for this modelling approach. Stochastic extensions of Petri nets [12, 13] had been proposed to overcome the problems of expressiveness and simultaneous resource possession, but they are not equipped with a natural compositionality to tackle complexity. Stochastic process algebras are extensions of classical process algebras such as CCS [14] and CSP [15], which are based on a compositional structure and a focus on the interaction or communication between components.

From the performance modelling perspective, classical process algebras lack quantitative information about the timing and likelihood of actions. The original stochastic process algebras (TIPP, EMPA, PEPA [16, 17, 18]) tackled this by associating an exponentially distributed random variable, representing duration, with each action. Later stochastic process algebras such as IMC and MoDeST [19, 20] separated immediate actions from exponentially distributed delays. In both cases it is straightforward to derive an underlying CTMC from the stochastic process algebra description.

2.1. PEPA

PEPA is a CSP-like process calculus extended with the notion of exponentially distributed activities [18]. A PEPA model consists of a collection of components (also termed processes) which undertake actions. A component may perform an action autonomously (*independent actions*) or in synchronisation with other components in the system (*shared actions*). The language supports the following operators:

Prefix $(\alpha, r).E$ constitutes the atomic unit of computation of a PEPA model. It denotes a component which may perform an activity (α, r) of type α, subsequently behaving as E, which is said to be a *derivative* of the component. The activity rate r is taken from the domain $\mathbb{R}_{>0} \cup \{\top\}$. If the rate is a positive real then the activity duration is assumed to be drawn from an exponential distribution with mean $1/r$ time units. The symbol $\top$ denotes a form of *passive synchronisation* whereby an activity of type α is to be executed in synchronisation with some other component, which will determine the overall rate of execution of the shared action. The models presented in this paper do not make use of passive synchronisation. The set of action types in a PEPA model is denoted by $\mathcal{A}$, whereas $\mathcal{A}ct$ denotes the set of activities.

Choice $E + F$ indicates that a component may behave as E or F. Unlike traditional process calculi in which the choice is non-deterministic, the behaviour in PEPA (and indeed in all other stochastic process calculi) is determined stochastically. For instance, let $r, s > 0$, in the choice $(\alpha, r).E + (\beta, s).F$ the actions α and β are executed with probabilities $r/(r+s)$ and $s/(r+s)$, respectively.

Constant $A \stackrel{def}{=} E$ is used to model cyclic behaviour. Consider $A \stackrel{def}{=} (\alpha, r).B$, $B \stackrel{def}{=} (\beta, s).A$. Here, A is a component with two derivatives which performs sequences of α- and β-activities forever.

Cooperation $E \underset{L}{\bowtie} F$ is the synchronisation operator of PEPA. The components E and F are required to synchronise over the action types in the set L. All the other actions are performed autonomously. For instance, $(\alpha, r).(\beta, s).E \underset{\{\alpha\}}{\bowtie} (\alpha, t).(\gamma, u).F$ is a cooperation between two components which may perform a shared activity of type α, with rate $\min(r, t)$, subsequently behaving as $(\beta, s).E \underset{\{\alpha\}}{\bowtie} (\gamma, u).F$. Then actions β and γ are carried out autonomously. By contrast, in the cooperation $(\alpha, r).E \underset{\{\alpha\}}{\bowtie} (\beta, s).F$ the process $(\alpha, r).E$ does not progress because α is not available in the right hand side of the cooperation. The set of all shared action types between E and F is sometimes denoted by the symbol $*$.

Hiding E/L relabels the activities of E with the *silent action* τ for all types in L. Thus, $\big((\alpha, r_1).E/\{\alpha\}\big) \underset{\{\alpha\}}{\bowtie} (\alpha, r_2).F$ does not cooperate over α because the process in the left-hand side of the cooperation performs a transition (τ, r_1) to E. All α-transitions performed by E are similarly hidden.

An interesting class of PEPA models comprises those which can be generated by the two-level grammar shown below.

$$S ::= (\alpha, r).S \quad | \quad S + S \quad | \quad A_S, \quad A_S \stackrel{def}{=} S$$

$$C ::= S \quad | \quad C \underset{L}{\bowtie} C \quad | \quad C/L \quad | \quad A_C, \quad A_C \stackrel{def}{=} C$$

The first production defines *sequential components*, i.e., processes which only exhibit sequential behaviour (by means of the prefix operator), and branching (by means of the choice operator). The second production defines *model components*, in which the interactions between the sequential components are expressed through the cooperation and hiding operators. The *system equation* designates the model component that defines the environment which embraces all of the behaviour of the system under study.

$$Client \stackrel{def}{=} (comm, r_d).Think$$
$$Think \stackrel{def}{=} (think, r_t).Client$$
$$Server \stackrel{def}{=} (comm, r_u).Log$$
$$Log \stackrel{def}{=} (log, r_l).Server$$
$$System_1 \stackrel{def}{=} Client[N_C] \underset{\{comm\}}{\bowtie} Server[N_S]$$

FIGURE 1. Simple PEPA model of a client/server scenario.

The model shown in Figure 1 is defined using such a grammar and will be used in the following to illustrate the main properties of PEPA. The model may represent a basic client/server scenario. A client is a sequential component which cycles between the two derivatives *Client* and *Think*. Similarly, a server is a two-derivative component with derivatives *Server* and *Log*. In derivative *Client* the client is able to carry out a shared action *comm* in cooperation with the server's derivative *Server*. The derivatives *Think* and *Log* model autonomous activities performed by the components. In a distributed application, activities of this kind may be used to denote genuinely local computations or to abstract away interactions with other components with negligible impact on the performance characteristics of the system. The system equation includes the derived syntax of a *component array* $E[N]$ which occurs frequently in the modelling of large-scale systems representing a population of N independent and identical sequential components E. More formally, $E[N]$ is a shorthand notation for $\underbrace{E \underset{\emptyset}{\bowtie} E \cdots \underset{\emptyset}{\bowtie} E}_{N}$.

3. SHIFTING LEVELS OF ABSTRACTION

Stochastic process algebras, such as PEPA, are typically given a semantics in terms of a labelled transition system, derived from small-step operational semantics. In other words, a set of semantic rules, shown in Figure 2, detail the possible evolutions of a term in the langage based on the syntactical construction of the term. The transitions which are derived are labelled by the activities and thus contain information about the dynamic behaviour in terms of the expected rate of the transition in addition to the type of activity performed. This inclusion of information about the rates within the labelled transition system means that a multi-transition system must be used in order to correctly reflect the dynamics of the system, i.e. if there are multiple instances of the same transition the resulting action will occur at a faster rate than if there is only a single instance, because each instance contributes to the apparent rate of the action.

Prefix

$$\mathrm{S}_0 : \frac{}{(\alpha, r).E \xrightarrow{(\alpha,r)} E}$$

Choice

$$\mathrm{S}_1 : \frac{E \xrightarrow{(\alpha,r)} E'}{E + F \xrightarrow{(\alpha,r)} E' + F}$$

$$\mathrm{S}_2 : \frac{F \xrightarrow{(\alpha,r)} F'}{E + F \xrightarrow{(\alpha,r)} E + F'}$$

Cooperation

$$\mathrm{C}_0 : \frac{E \xrightarrow{(\alpha,r)} E'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,r)} E' \underset{L}{\bowtie} F}, \alpha \notin L$$

$$\mathrm{C}_1 : \frac{F \xrightarrow{(\alpha,r)} F'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,r)} E \underset{L}{\bowtie} F'}, \alpha \notin L$$

$$\mathrm{C}_2 : \frac{E \xrightarrow{(\alpha,r_1)} E' \qquad F \xrightarrow{(\alpha,r_2)} F'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,R)} E' \underset{L}{\bowtie} F'}, \alpha \in L$$

$$R = \frac{r_1}{r_\alpha(E)} \frac{r_2}{r_\alpha(F)} \min\left(r_\alpha(E), r_\alpha(F)\right)$$

Hiding

$$\mathrm{H}_0 : \frac{E \xrightarrow{(\alpha,r)} E'}{E/L \xrightarrow{(\alpha,r)} E'/L}, \alpha \notin L$$

$$\mathrm{H}_1 : \frac{E \xrightarrow{(\alpha,r)} E'}{E/L \xrightarrow{(\tau,r)} E'/L}, \alpha \in L$$

Constant

$$\mathrm{A}_0 : \frac{E \xrightarrow{(\alpha,r)} E'}{A \xrightarrow{(\alpha,r)} E'}, A \stackrel{def}{=} E$$

FIGURE 2. Markovian semantics of PEPA.

The rules in Figure 2 correspond to the operators of the language introduced in the previous section. Most of the rules are straightforward, and presented here without comment. Rule C_2 is the fundamental inference for the characterisation of the dynamic behaviour of a shared action. It implements the semantics of *bounded capacity*: informally, the overall rate of execution of a shared activity is the minimum between the rates of the synchronising components. The rule relies on the notion of *apparent rate* to compute the total capacity of a cooperating component, according to the following definition.

The *apparent rate* of action α in process E, denoted by $r_\alpha(E)$, indicates the overall rate at which α can be

performed by E. It is recursively defined as:

$$r_\alpha\left((\beta, r).E\right) = \begin{cases} r & \text{if} \quad \beta = \alpha, \\ 0 & \text{if} \quad \beta \neq \alpha, \end{cases}$$

$$r_\alpha\left(E + F\right) = r_\alpha\left(E\right) + r_\alpha\left(F\right),$$

$$r_\alpha\left(E \underset{L}{\bowtie} F\right) = \begin{cases} \min\left(r_\alpha\left(E\right), r_\alpha\left(F\right)\right) & \text{if } \alpha \in L, \\ r_\alpha\left(E\right) + r_\alpha\left(F\right) & \text{if } \alpha \notin L, \end{cases}$$

$$r_\alpha\left(E/L\right) = \begin{cases} r_\alpha\left(E\right) & \text{if} \quad \alpha \notin L, \\ 0 & \text{if} \quad \alpha \in L. \end{cases}$$

According to this definition, for the array of sequential components $Client[N_C]$ the apparent rate of *comm* is

$$r_{comm}\left(Client[N_C]\right) = N_C \times r_{comm}\left(Client\right) = N_C \times r_d. \tag{1}$$

Similarly,

$$r_{comm}\left(Server[N_S]\right) = N_S \times r_{comm}\left(Server\right) = N_S \times r_u. \tag{2}$$

Once the labelled transition system, or *derivation graph*, corresponding to a PEPA model has been constructed then it can be interpreted as the state transition diagram of a CTMC. In this CTMC each state corresponds to a distinct syntactic form of the PEPA expression, as the model evolves according to the semantics. The CTMC is stored as an infinitesimal generator matrix, a matrix which captures the rates of transitions between states. From this the probability distribution over the states of the model at any given time, or at steady state, can be readily derived using standard linear algebra algorithms.

3.1. Identity and Individuality

The naive mapping to a CTMC outlined above is a very direct mapping of the process algebra description into a mathematical representation. In particular, if we consider PEPA components within the model, each is represented explicitly within the CTMC. Consider a system which is comprised of two copies of component P co-operating with component Q over activity α,

$$(P \parallel P) \underset{\{\alpha\}}{\bowtie} Q.$$

Suppose that components P and Q are specified as shown below.

$$P \stackrel{def}{=} (\alpha, r).P' \qquad Q \stackrel{def}{=} (\alpha, r).Q'$$

From the operational semantics of PEPA, the possible one-step derivatives of the model $(P \parallel P) \underset{\{\alpha\}}{\bowtie} Q$ are

$$(P' \parallel P) \underset{\{\alpha\}}{\bowtie} Q' \quad \text{and} \quad (P \parallel P') \underset{\{\alpha\}}{\bowtie} Q',$$

depending on whether the leftmost or the rightmost P participates in the cooperation with Q. In the naive mapping each of these states at the level of the process algebra semantics is represented by a distinct state in the underlying CTMC. Analysis of the CTMC derived in this way can distinguish states in which only the leftmost P has reached state P' from those in which only the rightmost P has reached state P'. Thus this interpretation of the model preserves both the *individuality* of the P components (each is treated separately) and the *identity*. Here we have dealt with identity informally, distinguishing the instances of P according to their position in the system equation. However we would obtain exactly the same CTMC if we were to write the model instead in terms of distinguished components with the same behaviour, e.g. $(P_1 \parallel P_2) \underset{\{\alpha\}}{\bowtie} Q$, where

$$P_1 \stackrel{def}{=} (\alpha, r).P_1', \quad P_2 \stackrel{def}{=} (\alpha, r).P_2', \quad \text{and} \quad Q \stackrel{def}{=} (\alpha, r).Q'.$$

This level of system description is typically useful in the early stages of model (and system) development, as it has sufficient detail to allow us to verify the correct operation of the model. This can be regarded as checking that the protocol which controls interactions between components is giving the desired behaviour. Due to the very explicit way in which all possible configurations of the process algebra model are represented in the underlying CTMC the state space based on such an interpretation will soon exceed the limits of efficient computation.

Fortunately, in most situations in which we are studying the performance of a system we are not interested in the identity of the components within the system. Once confidence has been gained in the correct operation of the model it is sufficient to know how many instances of a component are currently exhibiting a local state. In these circumstances we can disregard the identity of the instances within the system description although we still regard each instance as an individual. Taking this view of the system leads to a different mathematical interpretation. Again this will be a CTMC but one with a smaller state space as we would now represent both $(P' \parallel P) \underset{\{\alpha\}}{\bowtie} Q'$ and $(P \parallel P') \underset{\{\alpha\}}{\bowtie} Q'$ by a single state in the CTMC — the state which records that there is one instance of P in state P and one in state P'. The transition rates between the states in this CTMC capture the aggregated nature of the states represented. Formally we can define an equivalence relation at the level of the process algebra which identifies those process algebraic states which give rise to the same behaviour. If we denote the equivalence classes of states by $[E]$ where E is one PEPA expression exhibiting the behaviour we are interested in, we can observe that for our example

$$[(P \parallel P) \underset{\{\alpha\}}{\bowtie} Q] \xrightarrow{2\times r} [(P' \parallel P) \underset{\{\alpha\}}{\bowtie} Q'].$$

For PEPA such a reduction has been defined from both a theoretical perspective [21] and an algorithmic perspective [22] and is readily applied to models with repeated components. Note that the resulting mathematical representation preserves the individuality of components within the array but loses information about their identity within the array (i.e. their location within the expression).

The relationship between the CTMC obtained from the naive interpretation of the PEPA model, and that obtained using the aggregated interpretation based on counting has been extensively studied and is well-understood [22]. The two Markov processes are *lumpably equivalent*, meaning that the equivalence classes which are formed at the process algebra level, give rise to a *lumpable* partition of the naive CTMC. This has the consequence that the stochastic process in which each partition is a single state (the aggregated CTMC) is indeed a CTMC [23]. Moreover the performance indices derived at the level of the aggregated process can be readily be related back to the process algebra description in terms of individuals and identities [18, 21].

3.2. Collective Dynamics

The aggregated CTMC underlying a PEPA model is typically much more compact than the direct representation of the derivation graph in the naive CTMC. Nevertheless it can rapidly exceed the capabilities of current numerical solution tools, especially in circumstances when there are large numbers of instances of components within the model. In this situation it may be possible to switch to alternative means of analysis to derive properties from the CTMC. For example, stochastic simulation may be used to study the system, rather than numerical solution to find the probability distribution. This has the disadvantage that each run of the simulation generates only a single trajectory within the state space, without the consideration of all possible behaviours that are encompassed in the numerical solution. Thus simulation necessitates multiple runs of the model in order to derive statistically significant results [24]. However in situations when numerical solution becomes infeasible because the state space is too large, simulation's ability to avoid explicit representation of the entire state space is invaluable. The computational cost of analysis in this case remains high however.

In recent work we have been investigating how an alternative interpretation of the process algebra description of a system can lead to very efficient analysis based on a population level view of the components, instead of a view of the components as individuals [25]. In this view an array of components is treated as a single entity which undergoes change, as constrained by the structure of the interactions with other arrays of components (as specified by the system equation of the PEPA model) and the current state of that entity. Each entity/array of components captures a subpopulation within a system which is comprised of interacting subpopulations. A typical example might be that a population of clients will interact with a population of servers in a replicated service. Interaction tacks place between individuals but we can nevertheless observe many useful properties of the system by considering the consequences of these interactions at the population level. Therefore we can consider the performance of a large-scale system to be an emergent property of the collective dynamics of the individual components making up the system.

A crucial change when choosing this level of abstraction is that the *state* of the entity is now captured as a set of continuous variables rather than as discrete ones. Each subpopulation is represented by a set of variables, each variable representing one local derivative of the component. If this is done with discrete variables it gives rise to an equivalent representation to the aggregated CTMC: counting is used to record how many of each type of component are in each local state in both cases. An alternative set of structured operational semantic rules to generate a symbolic CTMC for such a case has recently been developed [25]. A significant computational benefit is gained when these variables are treated as continuous and their evolution is governed by a set of ordinary differential equations (ODEs) [26].

As an example, let us consider again $System_1$, as seen in Figure 1. This model has four local derivatives: *Client*, *Think*, *Server* and *Log*. In the discrete Markovian interpretation of the model states can be represented by a vector which uses non-negative integers to count the number of each type of local derivative at each state. For example, $(N_C, 0, N_S, 0)$ represents the initial state of the underlying CTMC, and $(N_C - 1, 1, N_S - 1, 1)$ is the state which is reached after an occurrence of the shared *comm* activity.

The continuous fluid approximation of the model also requires a system of four variables, but these will have *real* values, not integers. These ODE system variables are used to approximate the number of instances of each of the four local derivatives at any time point. The following association table shows the relationship between the ODE system variables and the processes of the PEPA model.

x_1	Approximate number of *Client* processes
x_2	Approximate number of *Think* processes
x_3	Approximate number of *Server* processes
x_4	Approximate number of *Log* processes

Starting from the PEPA definitions in Figure 1, the set of ODEs which is the fluid-flow approximation of the system behaviour can be derived algorithmically using the method presented in [25] and [26]. This is an entirely automatic process which runs without human

intervention and is implemented in the PEPA tools. By this process we obtain the following set of ODEs describing the evolution of the system:

$$\begin{aligned}
\frac{dx_1}{dt} &= -\min(x_1 \times r_d, x_3 \times r_u) + x_2 \times r_t, \\
\frac{dx_2}{dt} &= \min(x_1 \times r_d, x_3 \times r_u) - x_2 \times r_t, \\
\frac{dx_3}{dt} &= -\min(x_1 \times r_d, x_3 \times r_u) + x_3 \times r_l, \\
\frac{dx_4}{dt} &= \min(x_1 \times r_d, x_3 \times r_u) - x_3 \times r_l.
\end{aligned}$$

The minimum terms in the ODEs stem from PEPA's apparent rate rule, as defined in Section 3. Each of these is associated with the shared activity *comm* in the PEPA model. The ODE terms which do not use the minimum function arise from the individual activities in the PEPA model (*think* and *log*).

The PEPA model specifies the initial state of the system and this allows us to assign initial values to the ODE system variables thus: $x_1 = N_C$, $x_2 = 0$, $x_3 = N_S$ and $x_4 = 0$. This gives rise to a well-posed Initial Value Problem (IVP) where the evolution of the ODE system variables as a function of time can be obtained via numerical integration using well-known algorithms such as Runge-Kutta or Bulirsch-Stoer. The ODEs generated from PEPA models are conservative because processes are neither created nor destroyed in PEPA. The consequence of this for this model is that it will always be the case for all time points that $x_1 + x_2 = N_C$ and $x_3 + x_4 = N_S$.

This shift from the discrete, stochastic representation of a CTMC to the continuous, deterministic representation of a set of ODEs may seem surprising. The use of continuous variables is clearly an approximation since a variable of the system, for example the number of idle servers, or the number of occupied threads, will always be a natural number in reality. The events which impact on a variable such as the number of idle servers (e.g. the arrival of a customer) cause a discrete change in the system. Nevertheless, when subpopulations have large numbers of individuals we have found that the frequency and relative impact of the events mean that treating these changes as continuous is justified, at least empirically. Similarly, when subpopulations are large but follow the same pattern of behaviour, as defined in the PEPA description of the corresponding component, then the variability across the population is substantially reduced. In the limit, as the size of the population tends to infinity, then the behaviour of the stochastic process underlying the PEPA component tends to a deterministic limit [27, 28].

In [26] an algorithm was established to derive a set of ODEs representing the fluid approximation of a PEPA model. This can be regarded as an alternative interpretation of the system description in terms of populations and continuous variables. More recently a new symbolic semantics of PEPA models has been developed which allows the infinitesimal generator matrix underlying a PEPA model consisting of subpopulations to be constructed in a compact symbolic form [25]. In this form the number in the population is a dependent variable and it has been shown that for this CTMC as the population size tends to infinity the deterministic limit which is reached coincides with the ODEs derived directly from the PEPA model [25] (see Figure 3).

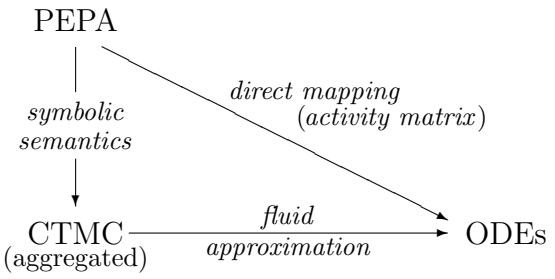

FIGURE 3. Alternative derivations from the PEPA description and the fluid limit

There is undoubtedly some loss of information as we move to the continuous, deterministic interpretation of the PEPA model but it nevertheless useful as it allows us to analyse models which would otherwise be infeasible. The sets of ODEs generated are rarely amenable to analytical solution but they are readily solved using numerical integration. Such analysis produces a time series of values for each of the continuous variables. Related to the conventional performance analyses conducted based on numerical solution of CTMCs these values correspond to the number of instances of each of the local states of the components of the model, which can be regarded as a form of utilisation. However we have also been able to develop rigorous methods to derive more sophisticated performance indices from the numerical integration of the ODEs [29, 30, 31], as will be illustrated in the following section.

4. CASE STUDY

The model which is presented in this section is based on the e-University case study of the SENSORIA project [32]. The case study is comprised of a number of scenarios; here the scenario of interest is the *Course Selection* scenario, where students obtain information about the courses available at their education establishment and may enrol in those for which specific requirements are satisfied.

Although the overall application is intended to be service-oriented, the scenario investigated here is such that the kinds of services available in the system do not to change over the time frame captured by this model. This reflects the fact that a university's course organisation is likely to be fixed before it is offered to

students. Furthermore, minor changes are likely not to affect the system's behaviour significantly. The model will not consider other services which may be deployed in an actual application (e.g. authentication services) because their impact on performance is assumed to be negligible. The scenario also considers a constant population of students to capture a real-world situation where the university's matriculation process is likely to be completed before the application may be accessed.

Our intention here is to give the reader an understanding of the modelling and analysis techniques used, rather than present novel or realistic results about web servers. Detailed models of web servers can be found in papers such as [33] and [34].

4.1. Model

The access point to the system is the *University Portal*, a front-end layer which presents the available services in a coherent way, for example by means of a web interface. There are four services in this model:

Course Browsing allows the user to navigate through the University's course offerings;

Course Selection allows the user to submit a tentative course plan which will be validated against the University's requirements and the student's curriculum;

Student Confirmation will force the student to check relevant personal details;

Course Registration will confirm the student's selection.

These components make use of an infrastructural *Database* service, which in turn maintains an event log through a separated *Logger* service.

The modelling paradigm adopted here captures the behaviour of a typical multi-threaded multi-processor environment used for the deployment and the execution of the application. The *University Portal* instantiates a pool of threads, each thread dealing with a request from a student for one of the services offered. During the processing of the request the thread cannot be acquired by further incoming requests, but when the request is fulfilled the thread clears its current state and becomes available to be acquired again. Analogous multi-threaded behaviour will be given to *Database* and *Logger*. Performance issues may arise from the contention of a limited number of threads by a potentially large population of students. If at some time point all threads are busy, further requests must queue, provoking delays and capacity saturation. This model also proposes another level of contention by explicitly modelling the processors on which the threads execute. Here, delays may occur when many threads try to acquire a limited number of processors available. Furthermore, this may be worsened by running several multi-threaded services on the same multi-processor system, as will be the case in the deployment scenario considered in this model: *University Portal* will run exclusively on multi-processor *PS*, whereas *Logger* and *Database* will share multi-processor *PD* (cf. Figure 4).

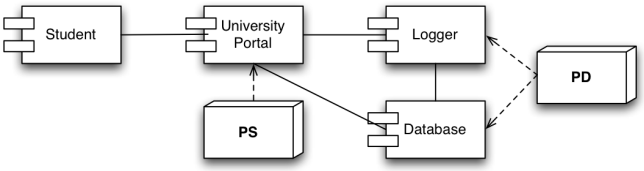

FIGURE 4. Deployment diagram of the e-University case study. Solid connectors between components indicate request/reply communication. Dashed lines denote the deployment of services onto processors.

4.1.1. General modelling patterns

Processing a request involves some computation on the processor on which the service is deployed. Such a computation in the PEPA model is associated with an activity $(type, rate)$, where *type* uniquely identifies the activity and *rate* denotes the average execution demand on the processor (i.e. $1/rate$ time units). A single processing unit may be modelled using a two-state sequential component. One state enables an *acq* activity to acquire exclusive access to the resource, while the other state enables all the activities deployed on the processor. Letting n be the number of distinct activities, the following pattern is used for a processor.

$$\begin{aligned} Processor_1 &\stackrel{def}{=} (acq, r_{acq}).Processor_2 \\ Processor_2 &\stackrel{def}{=} (type_1, r_1).Processor_1 \\ &\quad + (type_2, r_2).Processor_1 \\ &\quad + \ldots \\ &\quad + (type_n, r_n).Processor_1 \end{aligned} \tag{3}$$

This acquire-and-branch pattern describes a process which must first be acquired and then is used for exactly one job of the n possible types before being acquired again for the next job (possibly of a different type).

Communication in this model is synchronous and is modelled by a sequence of two activities in the form $(req_{from,to}, r_{req}).(reply_{from,to}, r_{rep})$ where the subscript *from* denotes the service from which the request originates and *to* indicates the service required. A recurring situation is a form of blocking experienced by the service invoking an external request. Let A and B model two distinct interacting services. For example,

$$A \stackrel{def}{=} (req_{A,B}, r_{reqA}).(reply_{A,B}, r_{repA}).A', \text{ and}$$
$$B \stackrel{def}{=} (req_{A,B}, r_{reqB}).(execute, r).(reply_{A,B}, r_{repB}).B'.$$

The communication between A and B will be expressed by means of the cooperation operator $A \underset{L}{\bowtie} B, L = \{req_{A,B}, reply_{A,B}\}$. According to the operational semantics, A and B may initially progress by

executing $req_{A,B}$, subsequently behaving as the process $(reply_{A,B}, r_{repA}).A' \underset{L}{\bowtie} (execute, r).(reply_{A,B}, r_{repB}).B'$. Now, although the left-hand side of the cooperation enables $reply_{A,B}$, the activity is not offered by the right-hand side, thus making the left-hand side effectively blocked until *execute* terminates (i.e., after an average duration of $1/r$ time units). These basic modelling patterns will be used extensively in this case study, as discussed next.

4.1.2. University Portal

A single thread of execution for the application layer *University Portal* is implemented as a sequential component which initially accepts requests for any of the services provided, as can be seen in the definitions below.

$$\begin{aligned} Portal \stackrel{def}{=} & (req_{student,browse}, \nu).Browse \\ + & (req_{student,select}, \nu).Select \\ + & (req_{student,confirm}, \nu).Confirm \\ + & (req_{student,register}, \nu).Register \end{aligned}$$

The rate ν will be used throughout this model in all the request/reply activities. In the following, the action type acq_{ps} is used to obtain exclusive access to processor PS.

Course Browsing is implemented as a service which maintains an internal cache. When a request is to be processed, the cache query takes $1/r_{cache}$ time units on average, and is successful with probability 0.95, after which the retrieved data is processed at rate r_{int}. Upon a cache miss, the information is retrieved by the *Database* service, and is subsequently processed at rate r_{ext}.

$$\begin{aligned} Browse &\stackrel{def}{=} (acq_{ps}, \nu).Cache \\ Cache &\stackrel{def}{=} (cache, 0.95 r_{cache}).Internal \\ &\quad + (cache, 0.05 r_{cache}).External \\ Internal &\stackrel{def}{=} (acq_{ps}, \nu).(internal, r_{int}).BrowseRep \\ External &\stackrel{def}{=} (req_{external,read}, \nu).(reply_{external,read}, \nu). \\ &\qquad (acq_{ps}, \nu).(external, r_{ext}).BrowseRep \\ BrowseRep &\stackrel{def}{=} (reply_{student,browse}, \nu).Portal \end{aligned} \tag{4}$$

Course Selection comprises four basic activities. An initial set-up task initialises the necessary data required for further processing (rate r_{prep}). Then, two activities are executed in parallel, and are concerned with validating the selection against the university requirements (rate r_{uni}) and the student's curriculum (rate r_{curr}), respectively. Finally, the outcome of this validation is prepared to be shown to the student (rate r_{disp}). The relative ordering of execution is maintained by considering three distinct sequential components. The first component prepares the data, then forks the two validating processes, waits for their completion, and finally displays the results.

$$\begin{aligned} Select &\stackrel{def}{=} (acq_{ps}, \nu).(prepare, r_{prep}).ForkPrepare \\ ForkPrepare &\stackrel{def}{=} (fork, \nu).JoinPrepare \\ JoinPrepare &\stackrel{def}{=} (join, \nu).Display \\ Display &\stackrel{def}{=} (acq_{ps}, \nu).(display, r_{disp}).SelectRep \\ SelectRep &\stackrel{def}{=} (reply_{student,select}, \nu).Portal \end{aligned}$$

The two validating processes are guarded by the fork/join barrier as shown below.

$$\begin{aligned} ValUni &\stackrel{def}{=} (fork, \nu).(acq_{ps}, \nu).(validate_{uni}, r_{uni}). \\ &\qquad (join, \nu).ValUni \\ ValCur &\stackrel{def}{=} (fork, \nu).(acq_{ps}, \nu).(validate_{cur}, r_{cur}). \\ &\qquad (join, \nu).ValCur \end{aligned} \tag{5}$$

These components will be arranged as follows in order to obtain a three-way synchronisation.

$$Select \underset{\{fork,join\}}{\bowtie} ValUni \underset{\{fork,join\}}{\bowtie} ValCur$$

Student Confirmation is represented in the PEPA model as an activity performed at rate r_{con}. The service uses the *Logger* component to register the event.

$$\begin{aligned} Confirm &\stackrel{def}{=} (acq_{ps}, \nu).(confirm, r_{con}).LogStudent \\ LogStudent &\stackrel{def}{=} (req_{confirm,log}, \nu). \\ &\qquad (reply_{confirm,log}, \nu).ReplyConfirm \\ ReplyConfirm &\stackrel{def}{=} (reply_{student,confirm}, \nu).Portal \end{aligned} \tag{6}$$

Finally, *Course Registration* performs some local computation (at rate r_{reg}) and then contacts the *Database* component to store the information.

$$\begin{aligned} Register &\stackrel{def}{=} (acq_{ps}, \nu).(register, r_{reg}).Store \\ Store &\stackrel{def}{=} (req_{register,write}, \nu). \\ &\qquad (reply_{register,write}, \nu).ReplyRegister \\ ReplyRegister &\stackrel{def}{=} (reply_{student,register}, \nu).Portal \end{aligned} \tag{7}$$

The "acquire-and-branch" pattern (3) is applied to processor PS to give the definitions shown below.

$$\begin{aligned} PS_1 &\stackrel{def}{=} (acq_{ps}, \nu).PS_2 \\ PS_2 &\stackrel{def}{=} (cache, r_{cache}).PS_1 + (internal, r_{int}).PS_1 \\ &\quad + (external, r_{ext}).PS_1 + (prepare, r_{prep}).PS_1 \\ &\quad + (display, r_{disp}).PS_1 + (validate_{uni}, r_{uni}).PS_1 \\ &\quad + (validate_{cur}, r_{cur}).PS_1 + (confirm, r_{con}).PS_1 \\ &\quad + (register, r_{reg}).PS_1 \end{aligned}$$

4.1.3. Database

This service exposes two functions for reading and writing data. Reading is a purely local computation, whereas writing additionally uses the *Logger* service. In this model, *Database* is only accessed by the university portal in states *External* and *Store* in equations (4) and (7), respectively. Let *PD* denote the processor on which *Database* is deployed, acquired through action $acquire_{pd}$. Similarly to *University Portal*, a single thread of execution for *Database* is defined by the following PEPA sequential component.

$$\begin{aligned}
Database &\stackrel{def}{=} (req_{external,read}, \nu).Read \\
&\quad + \ (req_{register,write}, \nu).Write \\
Read &\stackrel{def}{=} (acq_{pd}, \nu).(read, r_{read}).ReadReply \\
ReadReply &\stackrel{def}{=} (reply_{external,read}, \nu).Database \\
Write &\stackrel{def}{=} (acq_{pd}, \nu).(write, r_{write}).LogWrite \\
LogWrite &\stackrel{def}{=} (req_{database,log}, \nu). \\
&\quad (reply_{database,log}, \nu).WriteReply \\
WriteReply &\stackrel{def}{=} (reply_{register,write}, \nu).Database
\end{aligned} \tag{8}$$

4.1.4. Logger

This service accepts requests from *Student Confirmation* and *Database*, as described in equations (6) and (8), respectively. It is deployed on the same processor as *Database*, i.e., processor *PD*. Thus, one thread execution may be modelled as the PEPA sequential component shown below.

$$\begin{aligned}
Logger &\stackrel{def}{=} (req_{confirm,log}, \nu).LogConfirm \\
&\quad + \ (req_{database,log}, \nu).LogDatabase \\
LogConfirm &\stackrel{def}{=} (acq_{pd}, \nu). \\
&\quad (log_{conf}, r_{lgc}).ReplyConfirm \\
ReplyConfirm &\stackrel{def}{=} (reply_{confirm,log}, \nu).Logger \\
LogDatabase &\stackrel{def}{=} (acq_{pd}, \nu). \\
&\quad (log_{db}, r_{lgd}).ReplyDatabase \\
ReplyDatabase &\stackrel{def}{=} (reply_{database,log}, \nu).Logger
\end{aligned} \tag{9}$$

Taking together (8) and (9) it is possible to write a simple two-state PEPA sequential component that models the processor *PD*.

$$\begin{aligned}
PD_1 &\stackrel{def}{=} (acq_{pd}, \nu).PD_2 \\
PD_2 &\stackrel{def}{=} (read, r_{read}).PD_1 + (write, r_{write}).PD_1 \\
&\quad + \ (log_{conf}, r_{lgc}).PD_1 + (log_{db}, r_{lgd}).PD_1
\end{aligned}$$

4.1.5. Student Workload

A student is modelled as a sequential component which interacts with the university portal and accesses all of the services available. The behaviour is cyclic and the student interposes some think time between successive requests. This results in a *closed-workload* type of behaviour which is typical of many performance studies.

$$\begin{aligned}
StdThink &\stackrel{def}{=} (think, r_{think}).StdBrowse \\
StdBrowse &\stackrel{def}{=} (req_{student,browse}, \nu). \\
&\qquad (reply_{student,browse}, \nu).StdSelect \\
StdSelect &\stackrel{def}{=} (req_{student,select}, \nu). \\
&\qquad (reply_{student,select}, \nu).StdConfirm \\
StdConfirm &\stackrel{def}{=} (req_{student,confirm}, \nu). \\
&\qquad (reply_{student,confirm}, \nu).StdRegister \\
StdRegister &\stackrel{def}{=} (req_{student,register}, \nu). \\
&\qquad (reply_{student,register}, \nu).StdThink
\end{aligned}$$

4.1.6. System Equation

The multiplicity of threads and processors is captured in the system equation, in which all the sequential components illustrated above are composed with suitable cooperation operators to enforce synchronisation between shared actions. The complete system equation for this model is:

$$\begin{aligned}
&StdThink[N_S] \underset{*}{\bowtie} \\
&\quad \Big(\big(Portal[N_P] \underset{M_1}{\bowtie} ValUni[N_P] \underset{M_1}{\bowtie} ValCur[N_P] \big) \\
&\qquad \underset{M_2}{\bowtie} Database[N_D] \underset{M_3}{\bowtie} Logger[N_L] \Big) \underset{*}{\bowtie} \\
&\qquad\qquad \Big(PS_1[N_{PS}] \underset{\emptyset}{\bowtie} PD_1[N_{PD}] \Big),
\end{aligned}$$

where

$$\begin{aligned}
M_1 &= \{fork, join\} \\
M_2 &= \{req_{external,read}, reply_{external,read}, req_{register,write}, \\
&\qquad reply_{register,write}\} \\
M_3 &= \{req_{confirm,log}, reply_{confirm,log}, req_{database,log}, \\
&\qquad reply_{database,log}\}
\end{aligned}$$

and N_S, N_P, N_D, N_L, N_{PS} and N_{PD} are constants which specify the number of copies of each process. It is worth pointing out that the separate validating threads *ValUni* and *ValCur* inherit the multiplicity levels of the thread *Portal* which spawns them (i.e. N_P).

5. MODEL EVALUATION

This section is concerned with the analysis of the SENSORIA e-University case study. In Section 5.1 we consider the information about the system which can be gleaned from a qualitative interpretation of the PEPA model, disregarding timing information. The performance metrics of interest are discussed in Section 5.2. Section 5.3 illustrates the analysis of the system which can be based on the Markovian interpretation of the model, and Section 5.4 presents the results obtained with fluid-flow approximation.

N_S	N_P	N_D	N_L	N_{PS}	N_{PD}	*Size*
1	any	any	any	1	any	48
1	any	any	any	≥ 2	any	49
2	1	1	1	1	1	230
3	1	1	1	1	1	680
3	2	2	2	2	2	5540
10	2	2	2	2	2	512116
10	3	2	2	2	2	5075026

TABLE 1. State-space growth of the e-University case study.

5.1. Qualitative Analysis

As previously remarked, Markovian analysis is fundamentally limited by the rapid growth of the state space as a function of the population levels of the sequential components. Table 1 shows the state-space cardinality for some model configurations. Even for a small system with only ten clients (last row) the state space reaches over five million states; furthermore, there is a dramatic increase as a function of N_P (the number of portal threads)—adding one copy may result in an increase by a factor of ten (compare the last two rows). Nevertheless, analysis based on the explicit representation of the state space is a valuable tool for validating the correctness of the model. For example, the first two rows of Table 1 give confidence that the model matches the modeller's intended behaviour. Indeed, when there is only one student the state space is fairly small regardless of the multiplicity levels of threads and processors because at most only one of them will be used. Setting N_{PS} to any value greater than one adds only one more state because the two activities *ValCur* and *ValUni* of equation (5) may now run in parallel on two distinct processors (instead, when $N_{PS} = 1$ only one of them may access the same processor at a time). Another form of qualitative analysis can be based on visual inspection of the reachability graph, which can be walked through to generate possible trajectories of the system.

Further analysis may verify that the model is compliant with the policies of exclusive access to threads and processors. This analysis may be carried out by direct inspection of the state space and does not necessitate the solution of the underlying Markov chain. For instance, the configuration $N_S = 1, N_P = 2, N_D = N_L = N_{PS} = N_{PD} = 1$ considers a model with one student and two threads deployed for *University Portal*. Table 2 shows a subset of the state space in which each of the states enables one of the *University Portal*'s activities, i.e., *cache*, *prepare*, *confirm*, and *register*. The states are represented in a tabular form in which each column is associated with a local state of a sequential component. For the sake of conciseness not all sequential components are shown—in all cases, their local states are the initial ones. A necessary condition for the correctness of the model is that if one thread is engaged in some activity then the other must be idle, because at most one thread may be acquired at a time. This condition is met by all states of Table 2, since one *Portal* thread is always in its initial state. Incidentally, other behaviour seems to match the expected dynamics of the system. In particular, when a *Portal* thread is performing a *cache* action, the student is waiting for a reply to a browsing action, giving confidence that *cache* is indeed triggered by the shared $req_{student,browse}$ action. Similar considerations are valid for the local states of *Student* when the other activities are enabled. Furthermore, the fact that processor *PS* is in state PS_2 confirms that these activities are carried out in cooperation between *Portal* and *PS*.

Here we are analysing the model by inspection and informal consideration of the desirable behaviour. In practice it is often beneficial to use model checking in a tool such as PRISM [35] to test the validity of a logical expression of a system property with respect to the reachable states.

Exclusive access to processors may be checked in a similar manner. For example, the configuration $N_S = 2, N_P = 2, N_D = N_L = N_{PS} = N_{PD} = 1$ has now two students, each of whom may acquire one *Portal* thread. However, since there is only one processor on which the thread is running, there is contention at the thread/processor level. Exploring the complete state space of this system allows us to conclude that whenever one thread is using the processor, the other is either waiting for the processor to be released, or engaged in a communication. Specifically, if one thread enables one of the actions exhibited by PS_2 then the other thread is in one of its acq_{ps} states or undertaking a communication-related activity. We do not confirm this by showing the entire state space here due to its size (6970 states).

Explicit enumeration of the state space of a PEPA model is available in the Eclipse plug-in through a top-level menu item, as shown in Figure 5. The reachability graph may be iteratively walked using the *Single Step Navigator*, shown in Figure 6.

5.2. Metrics

Once confidence is gained that a model faithfully represents the system under consideration, the focus is shifted to performance evaluation. In this case study the system performance will be evaluated with respect to the *average response time* experienced by a student to carry out the complete sequence of operations with the university portal. The thinking time exhibited by the derivative *StdThink* will not be included as part of this response time. The performance is evaluated for steady-state conditions, i.e. after a sufficiently long time period that the system's state distribution does not change. Under these conditions, the computation of average response time in PEPA admits a simple

Student	*Portal*	*Portal*	*PS*
Action type *cache*			
$(reply_{student,browse}, \nu).StdSelect$	$Cache$	$Portal$	PS_2
$(reply_{student,browse}, \nu).StdSelect$	$Portal$	$Cache$	PS_2
Action type *prepare*			
$(reply_{student,select}, \nu).StdConfirm$	$Portal$	$(prepare,\ r_{prep}).ForkPrepare$	PS_2
$(reply_{student,select}, \nu).StdConfirm$	$(prepare,\ r_{prep}).ForkPrepare$	$Portal$	PS_2
Action type *confirm*			
$(reply_{student,confirm}, \nu).StdRegister$	$Portal$	$(confirm,\ r_{con}).LogStudent$	PS_2
$(reply_{student,confirm}, \nu).StdRegister$	$(confirm,\ r_{con}).LogStudent$	$Portal$	PS_2
Action type *register*			
$(reply_{student,register}, \nu).StdThink$	$(register,\ r_{reg}).Store$	$Portal$	PS_2
$(reply_{student,register}, \nu).StdThink$	$Portal$	$(register,\ r_{reg}).Store$	PS_2

TABLE 2. Tabular representation of a subset of the state space for the system configuration $N_S = 1, N_P = 2, N_D = N_L = N_{PS} = N_{PD} = 1$ showing that whenever a *Portal* thread is engaged in some action, the other is idle because one student may acquire at most one thread at a time.

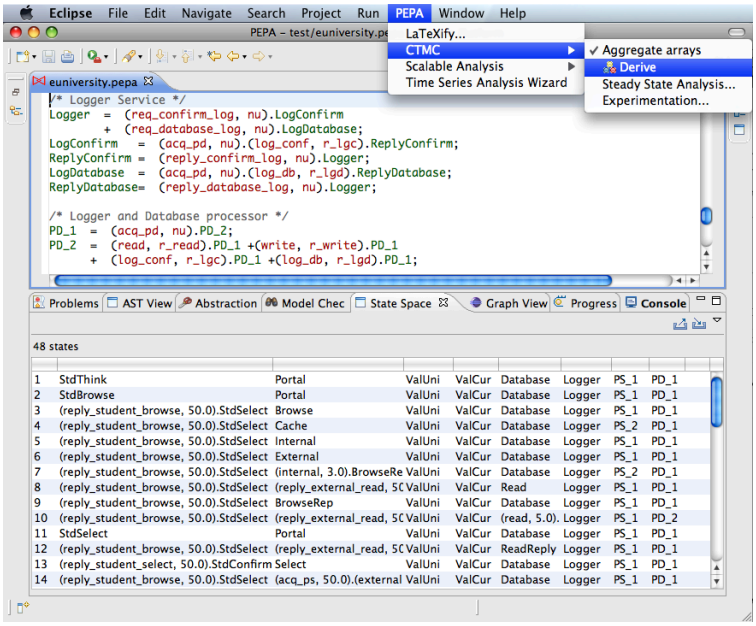

FIGURE 5. State-space exploration with the PEPA Eclipse plug-in. The top-level menu item *Derive* is available for any syntactically correct PEPA model. The *State Space View* (bottom) is updated with a tabular representation of the state space.

formulation based on Little's law [36], as discussed in [30]. Little's law says that in a system in a stationary state, the number of users L in the system is related to the throughput of user arrivals λ and the average response time W by the formula

$$L = \lambda W.$$

In this case study, L and λ can be computed in a straightforward way. The number of students in the system is equal to N_S (the total student population) minus the population of students who are thinking. This is directly obtained from the underlying differential equation model, since one coordinate, say $x_{StdThink}(t)$ (whose steady-state value will be denoted by $x_{StdThink}(\infty)$), is associated with the population count of the sequential component *StdThink*. The throughput of student arrival is given by the number of students performing the *think* action in the steady state. Since one single student carries out that action at rate r_{think}, the total throughput is the product $r_{think}\, x_{StdThink}(\infty)$. The average response time is therefore:

$$W = \frac{N_S - x_{StdThink}(\infty)}{r_{think}\, x_{StdThink}(\infty)}.$$

In practice, the average response time is calculated using the *PEPA Eclipse Plug-in*, a software tool which supports Markovian analysis and fluid-flow approximation of PEPA in the Eclipse framework [37].

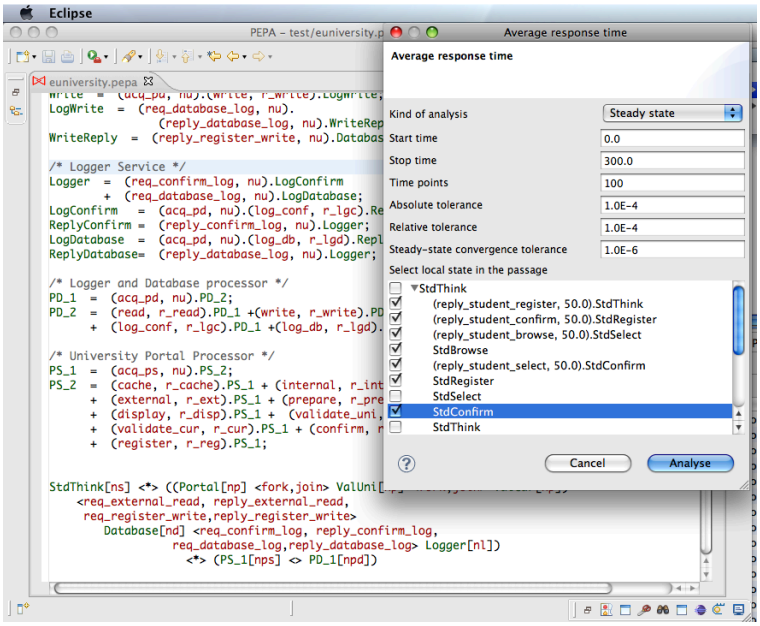

FIGURE 7. Screenshot of the PEPA Eclipse Plug-in showing the editor area (left) with an excerpt of the e-University case study and the dialogue box for the calculation of the average response time (right). The modeller is requested to set up the parameters for the ODE numerical integrator (top) and select which derivatives are to be interpreted as the user being in the system (bottom).

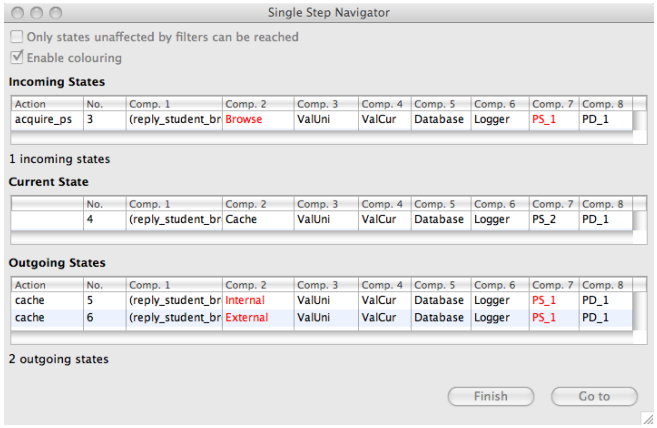

FIGURE 6. The *Single Step Navigator* allows the inspection of the reachability graph of a PEPA model. Given a configuration with all population counts set to one, this screenshot presents the neighbourhood of the state when the *cache* activity is being performed, showing that there are two possible outcomes leading to the local state *Internal* and *External*, respectively.

A screenshot of the tool is shown in Figure 7.

5.3. Markovian Analysis: Performance Bounds

Like the qualitative analysis considered earlier, the Markovian analysis is based on explicit enumeration of the state-space and is therefore limited to small-scale systems. Nevertheless carrying out a performance analysis of such small-scale systems can still offer valuable insight into the behaviour of the system. For example, here we show how a Markovian analysis can be used to derive some performance bound estimates for the e-University system.

For any given system configuration, the PEPA model obtained by setting $N_S = 1$ is optimal with respect to the performance perceived by the user, e.g., average response time. Indeed, a larger population of students cannot improve the performance because this would result in an increased contention for threads and processors. Markov chains underlying models with only one student are of very manageable size (cf. Table 1); therefore such bounds may be computed quickly and accurately. The system parameters used for this study (and throughout this section) are listed in Table 3. The first row of Table 4 shows the average response times as a function of the student population for a system configuration where all population counts of threads and processors are set to one. These results confirm that the average response time for $N_S = 1$ is indeed the minimum attainable for that configuration. Furthermore, in this particular model it is possible to conclude that the average response time at $N_S = 1$ for the configuration in the second row is a global minimum. In this case, the multiplicity of processors *PS* is set to two. This leads to an improved average response time because the two validating threads may now run effectively in parallel, as opposed to the previous case where they contend for the same processor. This configuration represents the maximum amount of resources needed by a single student, and the fact that the average response time does not increase further is confirmed, for instance, by the results shown

Messages	*Portal*	*Database*	*Logger*	*Student*
$\nu = 50.0$	$r_{cache} = 20$	$r_{read} = 5.0$	$r_{lgc} = 3.0$	$r_{think} = 0.08$
	$r_{int} = 3.0$	$r_{write} = 3.0$	$r_{lgd} = 3.5$	
	$r_{ext} = 4.0$			
	$r_{prep} = 5.0$			
	$r_{disp} = 8.0$			
	$r_{uni} = 5.0$			
	$r_{cur} = 4.0$			
	$r_{con} = 4.0$			
	$r_{reg} = 3.5$			

TABLE 3. Rate parameter set.

System Configuration								
N_P	N_D	N_L	N_{PD}	N_{PS}	$N_S{=}1$	$N_S{=}2$	$N_S{=}3$	$N_S = 4$
1	1	1	1	1	3.195	3.694	4.390	5.357
1	1	1	1	2	3.064	3.522	4.155	5.032
3	3	3	3	3	3.064	3.065	3.066	3.074

TABLE 4. Average response times calculated with small-sized CTMCs for the evaluation of performance bounds.

in the last row of the table, where all multiplicities are set to three.

5.4. Fluid-Flow Analysis: Scalability and Optimisation

To study the behaviour of the system under realistically sized user workloads, let us consider the following configuration: $N_P = N_D = N_L = 80, N_{PD} = 40, N_{PS} = 40$. The model gives rise to a set of 63 coupled ODEs, not shown in this paper for the sake of conciseness. Table 5 shows the results of fluid-flow analysis and stochastic simulation for the computation of the average response time for different student population sizes. The underlying ODE was solved using a fifth-order Range-Kutta numerical integrator. Stochastic simulation was conducted using the method of batch means which terminated when the 95% confidence interval was within 1% of the average. The accuracy, expressed as the absolute percentage relative error between the deterministic and the stochastic estimates, is adequate in all cases; importantly, the computational cost of fluid-flow analysis is confirmed to be negligible with respect to simulation, with runtimes separated by some orders of magnitude. For population levels between 1 and 325 the average response time is not impacted negatively, however further increases of the workload population cause a rather sharp degradation of the system performance (for instance, at $N_S = 600$ the average response time is about five times higher than its optimal level).

Due to its high effectiveness, fluid-flow analysis may be successfully employed in modelling situations

N_S	*ODE*	*Runtime*	*CTMC*	*Runtime*	*Error*
1	2.975	2.7 s	3.091	436 s	3.74%
300	2.975	2.7 s	3.105	2656 s	4.17%
325	2.975	2.6 s	3.329	3017 s	10.62%
350	3.686	2.9 s	3.863	6505 s	4.57%
400	5.999	7.3 s	5.993	4465 s	0.10%
500	10.623	6.6 s	10.534	3845 s	0.84%
600	15.248	6.6 s	15.233	2985 s	0.10%

TABLE 5. Scalability analysis: average response time as a function of the user workload. The results of fluid-flow analysis are compared against stochastic simulation of the Markov chain. Model configuration: $N_P = N_D = N_L = 80, N_{PD} = 40, N_{PS} = 40$. The table also shows the execution times for both methods.

Conf.	N_P	N_D	N_L	N_{PS}	N_{PD}	*Response time*
A	80	80	80	40	40	3.686
B	70	70	70	40	40	3.686
C	60	60	60	40	40	4.506
D	70	70	70	35	35	5.998
E	70	50	50	40	40	3.686
F	70	20	20	40	40	3.686
G	70	20	15	40	40	4.278
H	70	15	20	40	40	5.024

TABLE 6. Average response time for $N_S = 350$ and different system configurations.

which require the evaluation of the system under many different conditions. This problem is usually known as *capacity planning* and in the remainder of this section we examine one instance based on this case study. In addition to the workload population N_S, there are 20 other parameters in this model: 15 rate parameters and 5 concurrency levels for threads and processors. Let us suppose that the workload is known in advance and that the modeller has no possibility of intervention over the rate parameters (for instance, they may be determined by the technology infrastructure). In this scenario an interesting question is to determine an optimal configuration for the concurrency levels of the system. This is important because it is directly associated with the cost of deploying and running a service (fewer replicas may imply less memory or fewer processors required).

Let us consider a workload population of 350 students and suppose that the average response time of the system is acceptable (cfr. Table 5). Table 6 shows the response times calculated with different system configurations of similar size. Configuration A denotes the original system. Decreasing all thread multiplicities to 70 does not have any impact on the response time (compare A and B), however further reductions may incur some performance penalty (configuration C). Reducing the number of processors leads to more

significant delays (compare B and D). Comparing B with E and F, the user-perceived performance is not impacted negatively by decreasing the number of database and logger threads. Configurations G and H suggest that more parsimonious deployments cause noticeable performance degradation. Incidentally, they also show that the system is more sensitive to changes in the number of database threads than in the number of logger threads. In conclusion, F is the best configuration of those considered in Table 6, yielding the same performance but using 130 fewer threads than the original system. It must be pointed out that this strategy is not exhaustive, however it is not difficult to imagine the use of fluid-flow models in more sophisticated optimisation frameworks.

6. RELATED WORK

The first paper to develop a fluid-flow approximation to the dynamics of a stochastic process algebra was [26]. Early successes with this approach showed that it allowed modellers to create expressive models of complex large-scale systems [38, 39]. This focussed attention on the significance of the approach and motivated more deeply technical work on understanding the theoretical relationship between continuous and discrete PEPA models [28].

The very low evaluation cost of the fluid-flow approach enabled more extensive modelling studies to be carried out than had previously been possible. Thus, the SRMC calculus [40], based on PEPA, allows different system configurations to be expressed within the calculus itself, leading to a family of evaluation problems which are practical only because of the low unitary evaluation cost of fluid-flow models. This advantage is utilised to realise other extensive experimental programmes in papers such as [41, 42].

Process algebras are distinguished from other modelling formalisms because they have a formal language definition presented in the structured operational semantics style. Stochastic process algebras established the principle of deriving the underlying mathematical representation of the model via structured operational semantics [18]. Such an account was missing for the fluid-flow interpretation of PEPA until [25]. This put the continuous interpretation of PEPA on the same sure semantic foundations as the discrete interpretation and enabled it to be appreciated as an alternative semantic account of the language.

Some aspects of the PEPA language have proved difficult to interpret in the continuous domain. One such is the use of passive cooperation between concurrently active components. Here, some authors have studied the impact of different interpretations of passive cooperation on the numerical results computed by a PEPA model, considering methods of adjusting models in order to remove uses of passive cooperation [43, 44].

Having established the effectiveness of fluid-flow approximation numerically, and tested their accuracy empirically, attention is now being given to the definition of reward structures in the continuous domain such as response-time quantiles and other measures [31, 45].

Other authors have considered alternative approaches between the fully discrete and the fully continuous interpretation. A hybrid interpretation of PEPA, for application when not all components exist in sufficient quantities to justify a fluid approximation, was presented in [46]. Hybrid fluid-flow analysis with jump-diffusion SDEs has been considered by Hayden [47] and a Langevin interpretation of PEPA has been considered by Slegers [48]. Other languages such as HYPE [49] mix the discrete and the continuous domains within the language itself, in order to obtain a hybrid process calculus.

Beyond process algebras, fluid approximation has also been applied to stochastic Petri nets in both a pure and hybrid form, e.g. [50, 51]. Recent work on mean field approximation [52, 53, 54], giving rise to models expressed as systems of ODEs, is also closely related. This is particularly the case when, as in the context of PEPA, it is applied to models with continuous time [55]. This work can be seen to derive from earlier mean field techniques where models were approximated by a counting abstraction and approximation of state-dependent probabilities based on assuming mean conditions [56, 57]. The non-linear traffic equations of G-networks are mean-field equations which are exact rather than approximate [58, 59].

7. CONCLUSIONS

High-level modelling languages such as stochastic process algebras are built on rigorous mathematical foundations and make profitable use of powerful reasoning and analysis techniques. Building on discrete mathematical methods has considerable appeal because it naturally captures the discrete nature of computer software and computer data. However, discrete-state representations ultimately limit the applicability of modelling because of the problems of state-space growth. The potential for precise reasoning and analysis to improve systems then risks being lost because the techniques are perceived to be of limited applicability and appropriate only for small academic exercises.

Moving from discrete mathematical representations to continuous ones completely changes the nature of the problem. Considering populations instead of individuals bypasses the state-space explosion trap. The result is that modelling problems which were infeasible before are now not only feasible but they actually present little computational challenge. This has allowed researchers using these methods to apply precise numerical methods to large-scale systems with realistic population sizes.

REFERENCES

[1] Gilmore, S., Hillston, J., Holton, D., and Rettelbach, M. (1996) Specifications in Stochastic Process Algebra for a Robot Control Problem. *International Journal of Production Research*, **34**, 1065–1080.

[2] Holton, D. (1995) A PEPA specification of an industrial production cell. In Gilmore, S. and Hillston, J. (eds.), *Proceedings of the Third International Workshop on Process Algebras and Performance Modelling*, December, pp. 542–551. Special Issue of *The Computer Journal*, 38(7).

[3] Hermanns, H. and Katoen, J.-P. (2000) Automated compositional Markov chain generation for a plain-old telephone system. *Sci. Comput. Program.*, **36**, 97–127.

[4] Hillston, J. and Kloul, L. (2001) Performance investigation of an on-line auction system. *Concurrency and Computation: Practice and Experience*, **13**, 23–41.

[5] Bradley, J. T., Gilmore, S., and Hillston, J. (2008) Analysing distributed Internet worm attacks using continuous state-space approximation of process algebra models. *J. Comput. Syst. Sci.*, **74**, 1013–1032.

[6] Bravetti, M., Gilmore, S., Guidi, C., and Tribastone, M. (2008) Replicating web services for scalability. In Barthe, G. and Fournet, C. (eds.), *Proceedings of the Third International Conference on Trustworthy Global Computing (TGC'07)*, LNCS, **4912**, pp. 204–221. Springer-Verlag.

[7] Zhao, Y. and Thomas, N. (2008) Approximate solution of a PEPA model of a key distribution centre. In Kounev, S., Gorton, I., and 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, **5119**, pp. 44–57. Springer.

[8] Djoudi, L. and Kloul, L. (2008) Assembly code analysis using stochastic process algebra. In Thomas, N. and Juiz, C. (eds.), *Proceedings of the 5th European Performance Engineering Workshop (EPEW 2008)*, Palma de Mallorca, Spain, September, LNCS, **5261**, pp. 95–109. Springer.

[9] Massink, M., Harrison, M., and Latella, D. (2010) Scalable analysis of collective behaviour in smart service systems. *Proceedings of the 25th Annual ACM Symposium on Applied Computing*, pp. 1173–1180. ACM.

[10] Massink, M., Latella, D., Bracciali, A., and Harrison, M. (2010) A scalable fluid flow process algebraic approach to emergency egress analysis. *Proceedings of 8th IEEE International Conference on Software Engineering and Formal Methods*, pp. 169–180. Computer Society.

[11] Bell, A. and Haverkort, B. R. (2006) Distributed disk-based algorithms for model checking very large Markov chains. *Formal Methods in System Design*, **29**, 177–196.

[12] Molloy, M. (1981) On the integration of delay and throughput measures in distributed processing models. PhD thesis University of California, Los Angeles.

[13] Ajmone Marsan, M., Conte, G., and Balbo, G. (1984) A Class of Generalised Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems. *ACM Transactions on Computer Systems*, **2**, 93–122.

[14] Milner, R. (1989) *Communication and Concurrency*. Prentice-Hall.

[15] Hoare, C. (1985) *Communicating Sequential Processes*. Prentice-Hall.

[16] Götz, N., Herzog, U., and Rettelbach, M. (1992) TIPP—a language for timed processes and performance evaluation. Technical Report 4/92. IMMD7, University of Erlangen-Nürnberg, Germany.

[17] Bernardo, M. and Gorrieri, R. (1998) A tutorial on EMPA: A theory of concurrent processes with nondeterminism, priorities, probabilities and time. *Theoretical Computer Science*, **202**, 1–54.

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

[19] Hermanns, H. (2002) *Interactive Markov Chains: The Quest for Quantified Quality*, LNCS, **2428**. Springer.

[20] D'Argenio, P., Hermanns, H., Katoen, J.-P., and Klaren, R. (2001) Modest — a modelling and description language for stochastic timed systems. *Proc. of Process Algebra and Probabilistic Methods. Performance Modeling and Verification. Joint International Workshop, PAPM-PROBMIV 2001*, LNCS, **2165**. Springer-Verlag.

[21] Hillston, J. (1995) Compositional Markovian Modelling Using a Process Algebra. In Stewart, W. (ed.), *Numerical Solution of Markov Chains*. Kluwer.

[22] Gilmore, S., Hillston, J., and Ribaudo, M. (2001) An efficient algorithm for aggregating PEPA models. *IEEE Transactions on Software Engineering*, **27**, 449–464.

[23] Kemeny, J. and Snell, J. (1960) *Finite Markov Chains*. Van Nostrand.

[24] Mitrani, I. (1982) *Simulation techniques for discrete event system*. Cambridge University Press.

[25] Tribastone, M., Gilmore, S., and Hillston, J. (2010) Scalable differential analysis of process algebra models. *IEEE Transactions on Software Engineering*, –. IEEE Computer Society `http://doi.ieeecomputersociety.org/10.1109/TSE.2010.82`.

[26] Hillston, J. (2005) Fluid flow approximation of PEPA models. *Proceedings of the Second International Conference on the Quantitative Evaluation of Systems*, Torino, Italy, September, pp. 33–43. IEEE Computer Society Press.

[27] Kurtz, T. G. (1970) Solutions of ordinary differential equations as limits of pure jump Markov processes. *Journal of Applied Probability*, **7**, 49–58.

[28] Geisweiller, N., Hillston, J., and Stenico, M. (2008) Relating continuous and discrete PEPA models of signalling pathways. *Theor. Comput. Sci.*, **404**, 97–111.

[29] Tribastone, M. (2010) Scalable Analysis of Stochastic Process Algebra Models. PhD thesis The University of Edinburgh.

[30] Clark, A., Duguid, A., Gilmore, S., and Hillston, J. (2008) Espresso, a little coffee. *Proceedings of 7th Workshop on Process Algebra and Stochastically Timed Activities (PASTA 2008)*, Edinburgh, Scotland.

[31] Bradley, J. T., Hayden, R., Knottenbelt, W. J., and Suto, T. (2008) Extracting Response Times from Fluid

Analysis of Performance Models. *SIPEW'08, SPEC International Performance Evaluation Workshop, Darmstadt, 27-28 June 2008*, June, Lecture Notes in Computer Science, **5119**, pp. 29–43.

[32] Wirsing, M. and Hölzl, M. (eds.) (2011) *Rigorous Software Engineering for Service-Oriented Systems*. Springer-Verlag.

[33] Liu, X., Sha, L., Diao, Y., Froehlich, S., Hellerstein, J., and Parekh, S. (2003) Online response time optimization of Apache web server. In Jeffay, K., Stoica, I., and Wehrle, K. (eds.), *Proceedings of the 11th international conference on Quality of service (IWQoS'03)*, Heidelberg, pp. 461–478. Springer-Verlag.

[34] Do, T., Krieger, U., and Chakka, R. (2008) Performance modeling of an Apache web server with a dynamic pool of service processes. *Telecommunication Systems*, **39**, 117–129.

[35] Kwiatkowska, M., Norman, G., and Parker, D. (2009) PRISM: Probabilistic model checking for performance and reliability analysis. *ACM SIGMETRICS Performance Evaluation Review*, **36**, 40–45.

[36] Little, J. (1961) A proof for the queueing formula: $L = \lambda W$. *Operations Research*, **9**, 383–387.

[37] Tribastone, M., Duguid, A., and Gilmore, S. (2009) The PEPA Eclipse Plug-in. *Performance Evaluation Review*, **36**, 28–33.

[38] Duguid, A. (2006) Coping with the parallelism of BitTorrent: Conversion of PEPA to ODEs in dealing with state space explosion. In Asarin, E. and Bouyer, P. (eds.), *Formal Modeling and Analysis of Timed Systems, 4th International Conference, FORMATS 2006, Paris, France, September 25-27, 2006, Proceedings*, Lecture Notes in Computer Science, **4202**, pp. 156–170. Springer.

[39] Gilmore, S. and Tribastone, M. (2006) Evaluating the scalability of a web service-based distributed e-learning and course management system. In Bravetti, M., Núñez, M. T., and Zavattaro, G. (eds.), *Third International Workshop on Web Services and Formal Methods (WS-FM'06)*, Vienna, Austria, Lecture Notes in Computer Science, **4184**, pp. 156–170. Springer.

[40] Clark, A., Gilmore, S., and Tribastone, M. (2009) Scalable analysis of scalable systems. In Chechik, M. and Wirsing, M. (eds.), *Fundamental Approaches to Software Engineering, 12th International Conference, FASE 2009*, LNCS, **5503**, pp. 1–17. Springer.

[41] Zhao, Y. and Thomas, N. (2010) Efficient solutions of a pepa model of a key distribution centre. *Performance Evaluation*, **67**, 740 – 756. Special Issue on Software and Performance.

[42] Stefanek, A., Hayden, R., and Bradley, J. T. (2011) Fluid analysis of energy consumption using rewards in massively parallel Markov models. *International Conference on Performance Engineering (ICPE 2011)*, Karlsruhe, Germany.

[43] Hayden, R. A. and Bradley, J. T. (2010) Evaluating fluid semantics for passive stochastic process algebra cooperation. *Performance Evaluation*, **67**, 260 – 284. Performance Evaluation Methodologies and Tools: Selected Papers from VALUETOOLS 2008.

[44] Hayden, R. and Bradley, J. T. (2010) A fluid analysis framework for a Markovian process algebra. *Theoretical Computer Science*, **411**, 2260–2297.

[45] Tribastone, M., Ding, J., Gilmore, S., and Hillston, J. (2011) Fluid rewards for a stochastic process algebra. To appear in IEEE Transactions on Software Engineering.

[46] Bortolussi, L., Galpin, V., Hillston, J., and Tribastone, M. (2010) Hybrid semantics for PEPA. *QEST 2010, Seventh International Conference on the Quantitative Evaluation of Systems, Williamsburg, Viginia, USA, 15-18 September 2010*, pp. 181–190. IEEE Computer Society.

[47] Hayden, R. A. (2007). Addressing the state space explosion problem for PEPA models through fluid-flow approximation. Undergraduate project, Imperial College London.

[48] Slegers, J. (2010) A Langevin interpretation of PEPA models. *Electronic Notes in Theoretical Computer Science*, **261**, 71 – 89. Proceedings of the Fourth International Workshop on the Practical Application of Stochastic Modelling (PASM 2009).

[49] Galpin, V., Bortolussi, L., and Hillston, J. (2009) Hype: A process algebra for compositional flows and emergent behaviour. In Bravetti, M. and Zavattaro, G. (eds.), *CONCUR*, Lecture Notes in Computer Science, **5710**, pp. 305–320. Springer.

[50] David, R. and Alla, H. (2010) *Discrete, Continuous and Hybrid Petri Nets*, second edition. Springer.

[51] Gribaudo, M. and Telek, M. (2007) Fluid models in performance analysis. *Formal Methods for Performance Evaluation, 7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2007, Bertinoro, Italy, May 28-June 2, 2007, Advanced Lectures*, Lecture Notes in Computer Science, **4486**, pp. 271–317. Springer.

[52] Benaïm, M. and Boudec, J.-Y. L. (2008) A class of mean field interaction models for computer and communication systems. *Perform. Eval.*, **65**, 823–838.

[53] Bakhshi, R., Cloth, L., Fokkink, W., and Haverkort, B. (2009) Mean-field analysis for the evaluation of gossip protocols. *QEST*, pp. 247–256. IEEE Computer Society.

[54] Bakhshi, R., Endrullis, J., Endrullis, S., Fokkink, W., and Haverkort, B. (2010) Automating the mean-field method for large dynamic gossip networks. *QEST*, pp. 241–250. IEEE Computer Society.

[55] Bobbio, A., Gribaudo, M., and Telek, M. (2008) Analysis of large scale interacting systems by mean field method. *QEST*, pp. 215–224. IEEE Computer Society.

[56] Chesnais, A., Gelenbe, E., and Mitrani, I. (1983) On the modeling of parallel access to shared data. *Commun. ACM*, **26**, 196–202.

[57] Gelenbe, E. and Mitrani, I. (1982) Control policies in CSMA local area networks: Ethernet controls. *SIGMETRICS Performance Evaluation Review*, **11**, 233–240.

[58] Gelenbe, E. and Fourneau, J. (2002) G-Networks with resets. *Performance Evaluation*, **49**, 179–191.

[59] Fourneau, J. and Gelenbe, E. (2004) Flow equivalence and stochastic equivalence in G-Networks. *Computational Management Science*, **1**, 179–192.