# Stochastic simulation methods applied to a secure electronic voting model

Jeremy T. Bradley [1]

*Department of Computing*
*Imperial College London, England*

Stephen T. Gilmore [2]

*Laboratory for Foundations of Computer Science*
*The University of Edinburgh, Edinburgh, Scotland*

**Abstract**

We demonstrate a novel simulation technique for analysing large stochastic process algebra models, applying this to a secure electronic voting system example. By approximating the discrete state space of a PEPA model by a continuous equivalent, we can draw on rate equation simulation techniques from both chemical and biological modelling to avoid having to directly enumerate the huge state spaces involved. We use stochastic simulation techniques to provide traces of course-of-values time series representing the number of components in a particular state. Using such a technique we can get simulation results for models exceeding $10^{10000}$ states within only a few seconds.

## 1 Introduction

Voting is the foundation of the democratic process. All voting systems are fallible but electronic voting systems have had more profound problems than any of their enthusiasts could have anticipated. These have ranged from returning erroneous counts of votes to exposing the database of voters and votes to Internet access *during* the voting process on election day [1]. The net effect of these and a catalogue of other errors has been to undermine voter confidence in electronic voting systems [2].

When errors in computer systems become too problematic to ignore the classical solution in computer science is to address the problem by modelling

[1] Email: jb@doc.ic.ac.uk
[2] Email: stg@inf.ed.ac.uk

*This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science*
*URL:* www.elsevier.nl/locate/entcs

the systems in well-founded formal description languages and reasoning about these models. This confers the benefits of abstraction and clarification and sets the discourse on solid foundations. Such a clarification is obtained by concentrating on the *protocol* used to implement the secure voting process.

Methods such as theorem proving [3], model-checking [4] and static analysis [5] have been successful in discovering flaws in erroneous protocols and in providing strong guarantees of correct behaviour for sound protocols. The applications considered in [3,4,5] are concerned with the qualitative evaluation of secure protocols and are silent with regard to quantitative aspects. In the domain of electronic voting this is a noticeable omission: an election must be carried out in a timely manner.

Following [6], we consider quantitative analysis of a secure electronic voting protocol here. As in [6] we use Hillston's Performance Evaluation Process Algebra (PEPA) [7] as the formal language in which our models are expressed. PEPA is a well-known Markovian stochastic process algebra. Readers unfamiliar with PEPA are referred to Appendix A for an introduction and [7] for the formal definition of the language.

The meaning of models in the PEPA language is defined by an interleaving (small-step) structured operational semantics. An interleaving semantics interpretation of any process algebra model is prone to a potentially fatal problem: *state-space explosion*. The size of the state-space as a whole is bounded by the product of the sizes of the local state-space size of sequential components in the model. Even if a compact representation can be found for the state-space (for example, an aggregated representation [8] or an MTBBD encoding [9]), a stochastic process algebra model will additionally usually require the probability distribution over the state-space of the model to be stored. A compact representation of this vector cannot usually be found, even if one can be found for the state-space.

As noted in [6], numerical solution of the underlying CTMC of the secure electronic voting model cannot be conducted for voter populations of realistic size. Therefore, in order to make progress on this problem, where the issue of greatest significance in model appreciation is *scaling up*, we try a different approach: stochastic simulation.

Stochastic simulation methods developed in the domain of computational systems biology [10,11] and stochastic process algebras, such as Priami's stochastic $\pi$-calculus [12], have already been used to model biological systems and mapped to stochastic simulation models [13]. To the best of our knowledge, the present work is the first paper to apply the stochastic simulation methods known in computational systems biology to a stochastic process algebra performance model of a computing application. This is not an entirely simple change of domain, we have had to implement custom rate functions to model the behaviour of computational systems which have different interaction characteristics from biological ones.

## 2 Related work

There exist already a number of simulators for the PEPA language [14,15,16,17] so in this section we compare our novel stochastic simulator implementation with others.

The first simulator which was implemented for the PEPA language was Clark's PEPAroni simulator [14]. This was a prototype discrete-event simulator built on SimJava [18] which directly interpreted the PEPA model. It was used both to investigate original PEPA models and an experimental clock-based semantics for PEPA which used general distributions in place of exponentials [19]. The PEPAroni simulator recorded the states of the model as generated by simulation, together with the frequency of occurrence of the states for use in later model analysis, and to allow the results of the simulator to be compared against numerical solution of the Markov chain, if this solution was available.

The PEPAroni simulator was extended by Stathopoulos [15] and incorporated into the PEPA Workbench. The PEPAroni simulator proved to be a very useful tool to investigate general distributions but when applied to purely Markovian PEPA models user experience was poor. The simulator would run for a long time and still not generate all of the states of the model, even for very small models (less than one hundred states). It was always faster to solve the Markov chain than to simulate, even if computing a transient solution instead of the steady-state probability distribution.

Clark implemented another simulator for PEPA by including his PEPA model editor into the Möbius multi-paradigm multi-formalism modelling platform. The Möbius simulator uses evaluation strategies which are considerably more efficient than those used by SimJava under PEPAroni but again interprets models without the use of model aggregation before simulation begins.

An alternative implementation method was followed by Powell [17], who simulates a PEPA model by compiling the model into a Java application and executes it to gather statistics on the use of each activity in the model. The full state-space is not represented, which has benefits for model scalability. However, this has the attendant disadvantage that the performance results obtained are not directly comparable to the results obtained by numerical solution, if this can be done. Rather, derived results need to be computed from the numerical solution in order to compare with the simulation results, and hence validate the simulator. This implementation strategy has other drawbacks in that the limiting factor can be the number of simultaneous threads of execution which can be supported on the Java virtual machine beneath.

An alternative method for course-of-values analysis in computational biology is numerical integration of a representation of the model as a series of coupled ordinary differential equations (ODEs). This use of ODEs is known in biological modelling as a "deterministic simulation". Some of the same issues which arise here with respect to reaction kinetics arise also in the ODE setting, as discussed in [20].

## 3 Stochastic simulation

Stochastic process algebras are *compositional* modelling formalisms so it seems most appropriate to attempt to make use of the receptiveness to aggregation which is afforded by compositionality.

Stochastic simulation methods were developed to analyse chemical reactions involving large numbers of species with complex reaction kinetics. No attempt is made to track individuals at the molecular level: rather, concentrations are approximated by continuous variables. This provides compact descriptions of the current state of the simulation as a vector of real-valued variables measuring the quantity of each species.

When modelling chemical reactions a commonly-used reaction kinetics is *mass action*, where the reaction rate for a compound reaction is proportional to the product of the quantities of species involved. Mass action kinetics are not used in PEPA, which defines a concept of *apparent rate* to preserve the notion that an interacting component has a *bounded capacity* which it cannot exceed.

Where PEPA implementations such as the PEPA Workbench [21] and the Imperial PEPA Compiler (ipc) [22] use approximations to reduce the cost of apparent rate evaluation the onus is on the implementors to understand the impact of the approximations [23]. For this reason, when applying stochastic simulation methods to PEPA models we compute the rate of synchronised activities in a way which respects the PEPA semantics of apparent rate and bounded capacity. This rests on the use of functional expressions for aggregate rates [3], as explained in the following section.

This form of simulation is a significant paradigm shift from state-space-based model analysis approaches which track every state-change of every individual. In its use of continuous variables stochastic simulation also contrasts with discrete-event simulation which deals with discrete-valued representations of numbers of individuals. The closest comparison in traditional simulation approaches would perhaps be a Monte Carlo Markov Chain simulation of an aggregated representation of the model.

## 4 Methodology

In this section, we outline the methodology behind simulation generation from PEPA models through a short example. Below is a PEPA description of a simple multi-client and multi-server model:

[3] That is, expressing apparent rate as functions of component count and action rate.

$$\begin{aligned}
\mathit{Client} &\stackrel{\mathit{def}}{=} (\mathit{compute}, \top).\mathit{Client}_1\\
\mathit{Client}_1 &\stackrel{\mathit{def}}{=} (\mathit{delay}, \mu).\mathit{Client}\\
\mathit{Server} &\stackrel{\mathit{def}}{=} (\mathit{compute}, \lambda).\mathit{Server}_1\\
\mathit{Server}_1 &\stackrel{\mathit{def}}{=} (\mathit{recover}, \nu).\mathit{Server}
\end{aligned}$$

The system equation for the model is:

$$\underbrace{(\mathit{Client} \parallel \cdots \parallel \mathit{Client})}_{N} \underset{\{\mathit{compute}\}}{\bowtie} \underbrace{(\mathit{Server} \parallel \cdots \parallel \mathit{Server})}_{M}$$

This describes a *Client* that waits to place a *compute* action on a server and then sees a *delay* before attempting another *Server compute*. The *Server* controls the rate of *compute*, $\lambda$, in the interaction, before undergoing a *recover* phase. Only after the *recover* can the *Server* service another client, by enabling a *compute* action again.

The system equation describes how the system is composed together. There are $N$ parallel *Client* processes which cooperate on the *compute* action with $M$ parallel *Server* components.

### *4.1 Rate Equations*

We briefly introduce the notation for rate equations which are largely based on chemical reaction equations.

**Evolution,** $nA \xrightarrow{r} mB$**:** $n$ copies of reactant $A$ can evolve into $m$ copies of product $B$. This reaction is enabled once there are $n$ or copies of $A$ in the system. If there is more than one enabled reaction, then they compete to be the first to evolve (an implicit competitive choice).

**Combination,** $nA + mB$**:** This specifies multiple copies of $A$ and $B$ as either reactant or product. If this occurs on the left hand side of an evolution, then at least $n$ copies of $A$ and $m$ copies of $B$ are required to enable the reaction. This + operator is distinct from the + operator of PEPA: indeed, in PEPA terms, this would represent a cooperation between the $A$ components and the $B$ components. The multiplicity of $A$s and $B$s in the reactant, known as the stoichiometry of the reaction, can be used to represent a PEPA cooperation over a shared action, as in:

$$\underbrace{(A \underset{\{a\}}{\bowtie} \cdots \underset{\{a\}}{\bowtie} A)}_{n} \underset{\{a\}}{\bowtie} \underbrace{(B \underset{\{a\}}{\bowtie} \cdots \underset{\{a\}}{\bowtie} B)}_{m}$$

where all $n$ $A$ components and all $m$ $B$ components must simultaneously enable an $a$ action in order to progress the cooperation.

### 4.2 Simulation generation

We are aiming to generate a simulation file for input into the Dizzy tool [24], which is traditionally used to model chemical and biological rate reactions. In doing this we need to maintain the PEPA semantics which are gleaned from years of performance research into computational systems. Fortunately, the Dizzy tool gives us flexibility to migrate from the mass action semantics of the standard chemical modelling paradigm. This issue only affects the cooperative actions within a model, in this case it affects the overall rate of the *compute* action.

The procedure for generating a Dizzy simulation description goes as follows:

**Identify state-changing actions.** We have three such actions which modify the states of the components *Client* and *Server*: *delay*, *compute* and *recover*. These actions become the labels for the rate equations.

**Identify source/target component states.** For each action, identify the source/target states of the component that will be affected by that action occurring. For instance, the *delay* action has a source state $Client_1$ and a target state *Client*. That is to say, the occurrence of a *delay* action will decrease the number of components in state $Client_1$ and increase the number in state *Client*.

Where there is an interaction, we will have multiple source and possibly multiple target states. The source states for *compute* are *Client* and *Server*, the targets are $Client_1$ and $Server_1$. This means that components in both *Client* and *Server* states must exist before the *reaction* or interaction can take place. The result is components of state $Client_1$ and $Server_1$ in the same ratio.

**Calculate reaction rate.** For each action, we generate a reaction rate based on the number of components capable of performing that action. For instance for the action *delay*, if there are $n(Client_1)$ components in state $Client_1$ then the overall observed rate of action *delay* will be $n(Client_1)\mu$. Combining this source/target state extraction with the rate calculations, we get the equivalent rate equations of Figure 1.

$$
\begin{aligned}
Client + Server &\xrightarrow{\theta(n(Client))n(Server)\lambda} Client_1 + Server_1 \\
Client_1 &\xrightarrow{n(Client_1)\mu} Client \\
Server_1 &\xrightarrow{n(Server_1)\nu} Server \\
&\text{where } \theta(x) = 1 \text{ if } x > 0, \text{ else } 0.
\end{aligned}
$$

Fig. 1. The multi-server/multi-client PEPA example as a set of rate equations.

The explanation of $\theta(n(Client))n(Server)\lambda$ rate for the *compute* action in Figure 1 can be explained as follows.

```
// Initialisation of the number of components
Client = N;
Server = M;
Client_1 = 0;
Server_1 = 0;

// Rate equations
delay,
  Client_1 -> Client, [ Client_1 * mu ];

recover,
  Server_1 -> Server, [ Server_1 * nu ];

compute,
  Client + Server -> Client_1 + Server_1,
       [ theta(Client) * Server * lambda ];
```

Fig. 2. Dizzy file description of the multi-Client/multi-Server example

Consider $C$ clients utilising $S$ servers to execute the *compute* action. The overall rate of the synchronised *compute* activity, as defined by the PEPA semantics in terms of the apparent rate of *compute*, extracted from the cooperating clients and servers, is given by:

$$\begin{aligned} \min(C\top, S\lambda) &= \begin{cases} S\lambda & : C > 0 \\ 0 & : C = 0 \end{cases} \\ &= \theta(C)S\lambda \end{aligned} \tag{1}$$

Hence we can use the $\theta(\cdot)$ function on the number of clients to get the simplified expression of the standard min-formula. This captures the passive synchronisation in the model.

In general, we would apply the standard apparent rate formula, so in the active synchronisation case, we would use a combined rate function of $\min(C\alpha, S\beta)$, for $C$ clients cooperating with $S$ servers at rates $\alpha$ and $\beta$ respectively.

### *4.3 Dizzy format*

Having obtained our rate equations for the individual actions of the PEPA model example, it is a straightforward process to turn these into the Dizzy file. This transformation is implemented in the PEPA Workbench. The resulting file is shown in Figure 2. In the Dizzy format, all the rate equations are labelled by the action name. The number of components in a given state $n(\mathtt{X})$ is given by `X` in the initialisation block and the rate description. The rate description, itself, is given in square brackets `[]`.

# 5 PEPA model

In this section we present a simulation model of the voting protocol expressed in PEPA. There are a number of significant differences from the model of [6].

(i) We model only one round of the election because we are conducting a course-of-values time series simulation instead of performing a steady-state computation. In [6] the voting process is made to cycle in order that the model defines an ergodic Markov chain. Here we have components which conduct their designated activities and then terminate. We use the definition of a terminated process in PEPA (denoted by **Stop**) from [25].

Thus the termination state of this model is an untidy one, as determined by the end point of the election: some voters may not ever register, some might not confirm that their votes were correctly recorded, and so forth. This contrasts with the requirement for tidy termination in order that the system is irreducible or strongly-connected (required in [6] for meaningful steady-state computation).

(ii) In contrast to [6] we use an inversion of control model to have a control process determining the progress of the election from one stage to the next. This leads to a simplification of the descriptions of the voters, administrators, collectors and counters in the model. Choices are removed from the definitions of these components and moved into the control process at the meta-level.

Thus, the two PEPA models are not in a relationship such as the bisimulation relation of *strong equivalence* [7] and are instead only alternative models of the same system.

## *5.1 Voting in the election*

We do not offer a full description of the voting scheme used here, but instead provide only an outline and refer the interested reader to [26] or [6].

Electronic voting can be divided into a preparation phase which is ended by contacting the administrator, voting which ends by contacting the collecting officer, and checking which may or may not lead to an appeal.

In the preparation phase the voter's activities include choosing the voting strategy and commiting to it using a bit commitment protocol. Blinding is used to ensure anonymity of ballots and digital signatures are used to ensure authentication.

The blinded, signed ballot is sent to an administrator for checking, and returned verified. The voter unblinds this and checks the signature. The last activity in the voting phase is to send the ballot to the collecting officer.

Vote counting begins, and ends when the vote counters publish a list of votes. A voter might appeal at this stage if their vote does not appear on the list.

$$
\begin{aligned}
Voter0 &\stackrel{def}{=} (choose, c_1).Voter0\_1\\
Voter0\_1 &\stackrel{def}{=} (bitcommit, b_1).Voter0\_2\\
Voter0\_2 &\stackrel{def}{=} (blind_1, b_2).Voter0\_3\\
Voter0\_3 &\stackrel{def}{=} (blind_2, b_3).Voter0\_4\\
Voter0\_4 &\stackrel{def}{=} (voter\_sign, s_1).Voter0\_5\\
Voter0\_5 &\stackrel{def}{=} (sendA, s_2).Voter0\_5b\\
Voter0\_5\mathrm{b} &\stackrel{def}{=} (sendV, \top).Voter1\\
Voter1 &\stackrel{def}{=} (unblind_1, u_1).Voter1\_1\\
Voter1\_1 &\stackrel{def}{=} (unblind_2, u_2).Voter1\_2\\
Voter1\_2 &\stackrel{def}{=} (verify_1, v_2).Voter1\_3\\
Voter1\_3 &\stackrel{def}{=} (verify_2, v_3).Voter1\_4\\
Voter1\_4 &\stackrel{def}{=} (sendC, s_6).Voter2\\
Voter2 &\stackrel{def}{=} (check, p \times c_4).Voter3\\
&+ \; (check, (1-p) \times c_4).Voter2b\\
Voter2\mathrm{b} &\stackrel{def}{=} (sendCo, s_7).Voter\_Finished\\
Voter3 &\stackrel{def}{=} (appeal, a_1).Voter2b\\
Voter\_Finished &\stackrel{def}{=} \mathbf{Stop}
\end{aligned}
$$

### 5.2 *The role of the administrator*

The adminstrator becomes active once the voter has registered, and takes them through to the point where they are able to cast their vote. This involves checking and verification of eligibility to vote, followed by digital signing of the ballot. The administrator finally sends the blinded ballot back to the voter.

$$
\begin{aligned}
Administrator &\stackrel{def}{=} (sendA, \top).Administrator\_2\\
Administrator\_2 &\stackrel{def}{=} (check, c_2).Administrator\_3\\
Administrator\_3 &\stackrel{def}{=} (check_2, c_3).Administrator\_4\\
Administrator\_4 &\stackrel{def}{=} (verify, v_1).Administrator\_5\\
Administrator\_5 &\stackrel{def}{=} (admin\_sign_1, s_3).Administrator\_6\\
Administrator\_6 &\stackrel{def}{=} (admin\_sign_2, s_4).Administrator\_7\\
Administrator\_7 &\stackrel{def}{=} (sendV, s_5).Administrator\_Finished\\
Administrator\_Finished &\stackrel{def}{=} \mathbf{Stop}
\end{aligned}
$$

### 5.3 Collection of the votes

Votes are received by a collecting officer. Their role is in the voting phase to check that the ballot has been correctly signed by an administrator. If this is verified then the collecting officer adds the vote to a list, labelling this with a unique reference number. This list will be published when the collecting period is over.

$$\begin{aligned}
\mathit{Collector_0} &\stackrel{def}{=} (\mathit{sendC}, \top).\mathit{Collector_0a} \\
\mathit{Collector_}0\mathrm{a} &\stackrel{def}{=} (\mathit{collector_verify}_1, v_4).\mathit{Collector_0a1} \\
\mathit{Collector_}0\mathrm{a1} &\stackrel{def}{=} (\mathit{collector_verify}_2, v_5).\mathit{Collector_0a2} \\
\mathit{Collector_}0\mathrm{a2} &\stackrel{def}{=} (\mathit{add}, a_2).\mathit{Collector_Finished} \\
\mathit{Collector_Finished} &\stackrel{def}{=} \mathbf{Stop}
\end{aligned}$$

### 5.4 Vote counting

The responsibility is placed with those counting votes to check that the strategy chosen by the voter in the first stage of the election process is a valid one and to make all cast votes ready for the final election count which ends the election.

$$\begin{aligned}
\mathit{Counter_1} &\stackrel{def}{=} (\mathit{sendCo}, \top).\mathit{Counter_1a} \\
\mathit{Counter_}1\mathrm{a} &\stackrel{def}{=} (\mathit{check_strategy}, c_5).\mathit{Counter_Finished} \\
\mathit{Counter_Finished} &\stackrel{def}{=} \mathbf{Stop}
\end{aligned}$$

### 5.5 The election process

The Election process itself is of a different character to the others in the model. The election itself is not an actor in the electoral process: rather it exists at the level of a virtual process controlling phases of the simulation, it could be considered as being part of the legal framework of the election. There is a similarity both with the net structure in a PEPA net [27] and with the *stochastic probes* [28] used to witness events in a PEPA model, but the control process is different from either in that it structures the voting process into phases (preparation, voting, counting, and finished), allowing selected activities in each phase, and prohibiting them where they are inappropriate.

A stochastic probe observes performance-significant events. A meta-level control process allows performance-significant events and generates simulation-control events (ending one phase, beginning another, and terminating the simulation overall).

It would be possible to realise the same effect in an alternative way using PEPA extended with *functional rates* [29]. The election process would be a function over the global state space of the model, allowing the appropriate actions at the appropriate times and disallowing them otherwise. We have chosen here to represent this function instead as a PEPA component and observe that the $\theta$ function would be a very suitable way in general to implement functional rates.

$$
\begin{array}{rcl}
\mathit{Election\_Preparation} & \stackrel{\mathit{def}}{=} & (\mathit{choose}, \top).\mathit{Election\_Preparation} \\
& + & (\mathit{bitcommit}, \top).\mathit{Election\_Preparation} \\
& + & (\mathit{blind}_1, \top).\mathit{Election\_Preparation} \\
& + & (\mathit{blind}_2, \top).\mathit{Election\_Preparation} \\
& + & (\mathit{voter\_sign}, \top).\mathit{Election\_Preparation} \\
& + & (\mathit{sendA}, \top).\mathit{Election\_Preparation} \\
& + & (\mathit{check}, \top).\mathit{Election\_Preparation} \\
& + & (\mathit{check}_2, \top).\mathit{Election\_Preparation} \\
& + & (\mathit{verify}, \top).\mathit{Election\_Preparation} \\
& + & (\mathit{admin\_sign}_1, \top).\mathit{Election\_Preparation} \\
& + & (\mathit{admin\_sign}_2, \top).\mathit{Election\_Preparation} \\
& + & (\mathit{sendV}, \top).\mathit{Election\_Preparation} \\
& + & (\mathit{publishA}, \mathit{er}).\mathit{Election\_Voting} \\
\mathit{Election\_Voting} & \stackrel{\mathit{def}}{=} & (\mathit{unblind}_1, \top).\mathit{Election\_Voting} \\
& + & (\mathit{unblind}_2, \top).\mathit{Election\_Voting} \\
& + & (\mathit{verify}_1, \top).\mathit{Election\_Voting} \\
& + & (\mathit{verify}_2, \top).\mathit{Election\_Voting} \\
& + & (\mathit{sendC}, \top).\mathit{Election\_Voting} \\
& + & (\mathit{collector\_verify}_1, \top).\mathit{Election\_Voting} \\
& + & (\mathit{collector\_verify}_2, \top).\mathit{Election\_Voting} \\
& + & (\mathit{add}, \top).\mathit{Election\_Voting} \\
& + & (\mathit{publishC}, \mathit{er}).\mathit{Election\_Counting} \\
\mathit{Election\_Counting} & \stackrel{\mathit{def}}{=} & (\mathit{check}, \top).\mathit{Election\_Counting} \\
& + & (\mathit{check}, \top).\mathit{Election\_Counting} \\
& + & (\mathit{sendCo}, \top).\mathit{Election\_Counting} \\
& + & (\mathit{appeal}, \top).\mathit{Election\_Counting} \\
& + & (\mathit{check\_strategy}, \top).\mathit{Election\_Counting} \\
& + & (\mathit{final\_publish}, \mathit{er}).\mathit{Election\_Finished} \\
\mathit{Election\_Finished} & \stackrel{\mathit{def}}{=} & \mathbf{Stop}
\end{array}
$$

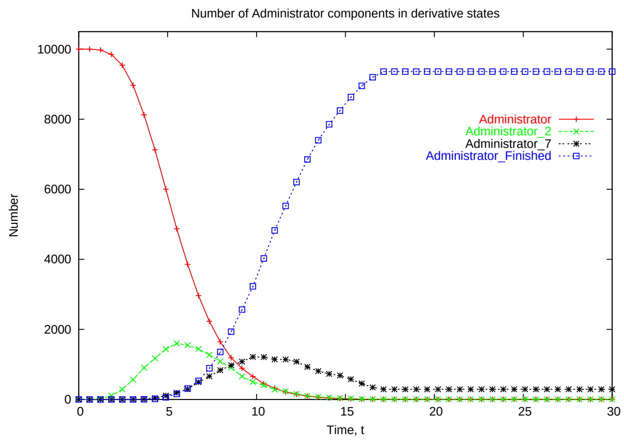


Fig. 3. A simulation of the Administrator component

The system which we analysed was composed of the above sequential components in the following assembly:

$$Election\_Preparation \underset{\mathcal{L}}{\bowtie} Electoral\_Personae$$

where:

$$Electoral\_Personae \stackrel{def}{=} Voter_0[N] \underset{\mathcal{M}}{\bowtie} Electoral\_Apparatus$$
$$Electoral\_Apparatus \stackrel{def}{=} Collector_0[N] \mid\mid Counter_1[N] \mid\mid Administrator[N]$$

and:

$$N = 10,000$$
$$\mathcal{L} = \{choose, bitcommit, blind_1, blind_2, voter\_sign, sendA, sendV, unblind_1, unblind_2, verify_1, verify_2, sendC, check, check, sendCo, appeal, publishA, check, check_2, verify, admin\_sign_1, admin\_sign_2, collector\_verify_1, collector\_verify_2, add, publishC, check\_strategy, final\_publish\}$$
$$\mathcal{M} = \{sendA, sendV, sendC, sendCo, publishC\}$$

### *5.6 Simulation results*

Figures 3 to 9 show information extracted from simulations of the voting model. In each case, the numbers of derivatives of a component (possible successor states of a component) are shown against time. So as not to over-clutter the diagrams, we have only shown qualitatively distinct derivative traces.

In Figure 3, we present a selection of simulations for different derivatives of the Administrator component. The first component plot is of the number of Administrator components which have not seen a transition *sendA* out of the Administrator state. There is a slight delay while the Administrators wait to synchronise with the first *sendA* actions from the population of Voters, but thereafter the decline in number is almost exponential. The derivatives

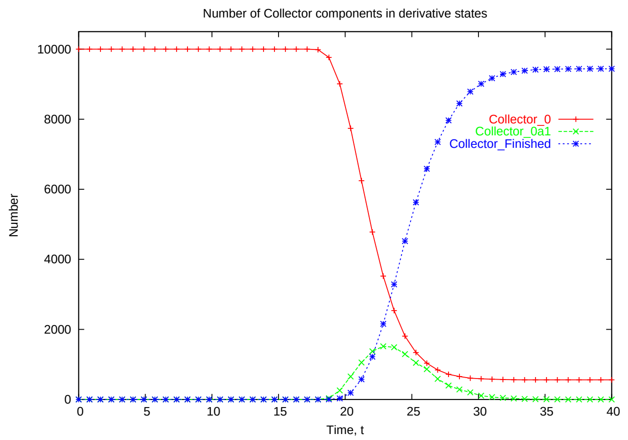


Fig. 4. A simulation of the Collector component

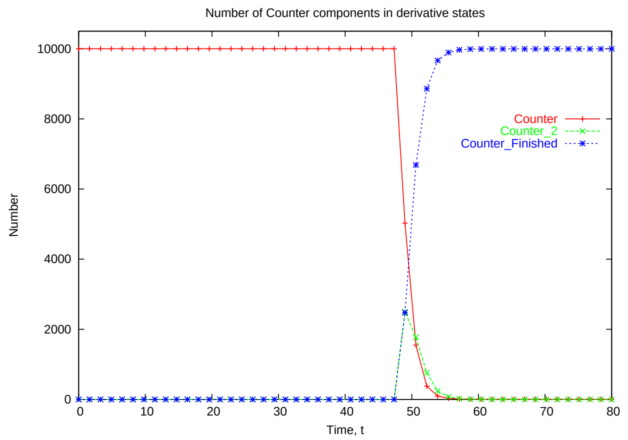


Fig. 5. A simulation of the Counter component

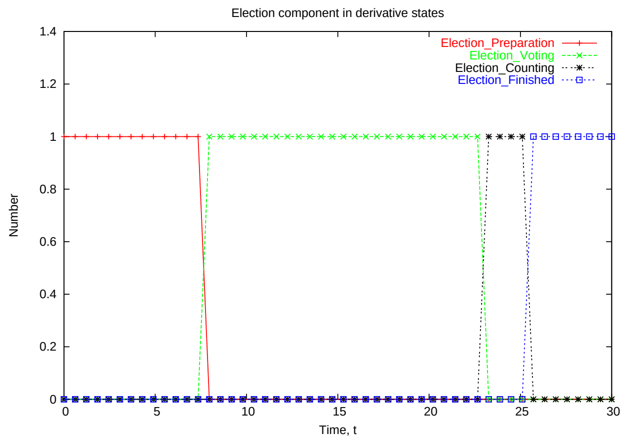


Fig. 6. A simulation of the Election component

$Administrator_2$ and $Administrator_7$ are transient states of the component and so the populations here almost approach 0. The last state and also the absorb-

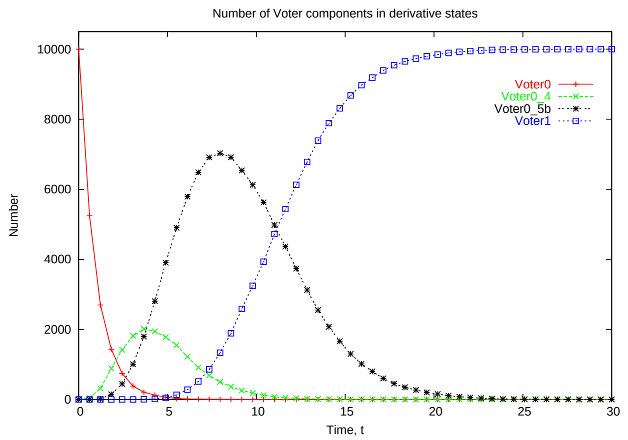


Fig. 7. A simulation of the Voter component through its early evolution to *Voter*1

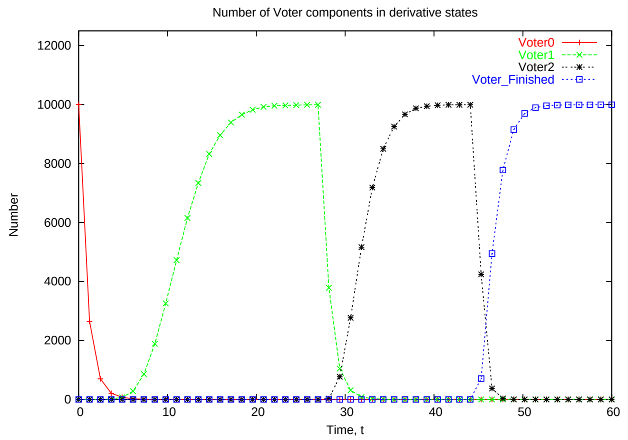


Fig. 8. A simulation of the different phases of the Voter component from *Voter*0 to *Voter_Finished*

ing state of the component is *Administrator_Finished*, which ends up with the bulk of the population in this trace.

Figure 4 traces the number of Collector component derivatives over time sees similar population dynamics to that of the Administrator. The only difference being a much longer delay, in this case, before the initial $Collector_0$ state is left. This is down to the *sendC* action which is seen much later on in the Voter lifecycle and initiates the Collector process.

Similarly, Figure 5 traces the number of Counter component derivatives over time. The only point of interest here is the very sharp decline in the trace of $Counter_0$ components which is unlike the smooth decline of previous components. We attribute this to the Election component shown in Figure 6, which controls the phases of the election and has a square-tooth profile. Since the *sendCo* action which sends the vote to the Counter and initiates the counting process is closely allied to the initiation of the last phase of the Election component, we see a replication of this sharp derivative change in the Counter

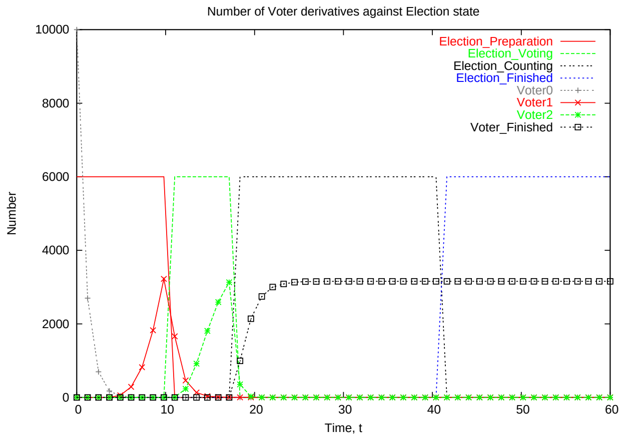


Fig. 9. A joint simulation of *Voter* and *Election* components where the phases of the *Voter* follow those of the *Election*

component as well.

The simulations of the Voter component are shown in Figures 7 and 8. Figure 7 shows the smooth evolution of *Voter* to derivative $Voter_1$. Figure 8 shows just the key derivatives through the whole evolution from *Voter* to *Voter_Finished*. Again the sharp derivative changes at the end of the plots for derivatives $Voter_1$ and $Voter_2$ are due to the synchronisation with the controlling Election component.

This close relationship between *Election* and *Voter* can be seen more closely in Figure 9, which shows both *Voter* and *Election* derivatives in the same simulation. Clearly, the termination of the $Voter_1$ and $Voter_2$ phases is attributed to the time-out for that phase of the election as dictated by the *Election* component, in its state change to *Election_Voting* and *Election_Counting* respectively. The end of the final Election phase is not seen by the Voter as it concerns the completion of counting the votes.

## 6 Implementation

We implemented our novel stochastic simulator for PEPA by extending the open-source Dizzy simulation and analysis platform [24] with our pre-existing analysis tool for PEPA, the PEPA Workbench [21]. Dizzy is implemented as a Java application with a graphical user interface whereas the PEPA Workbench is a command-line tool implemented in the functional programming language Standard ML. We brought these two tools together in an unusual way, by compiling the ML edition of the PEPA Workbench to Java bytecodes using the MLj compiler, which targets the JVM.

Dizzy has its own modelling language, and supports other modelling languages also, including the well-known Systems Biology Markup Language [30]. By adding the PEPA Workbench to Dizzy it is now possible to evaluate PEPA models on Dizzy and, because there exists a mapping from UML to PEPA [31], it is also possible to model with UML and use Dizzy as an analysis tool. A schematic of the extended Dizzy platform is shown in Figure 10.

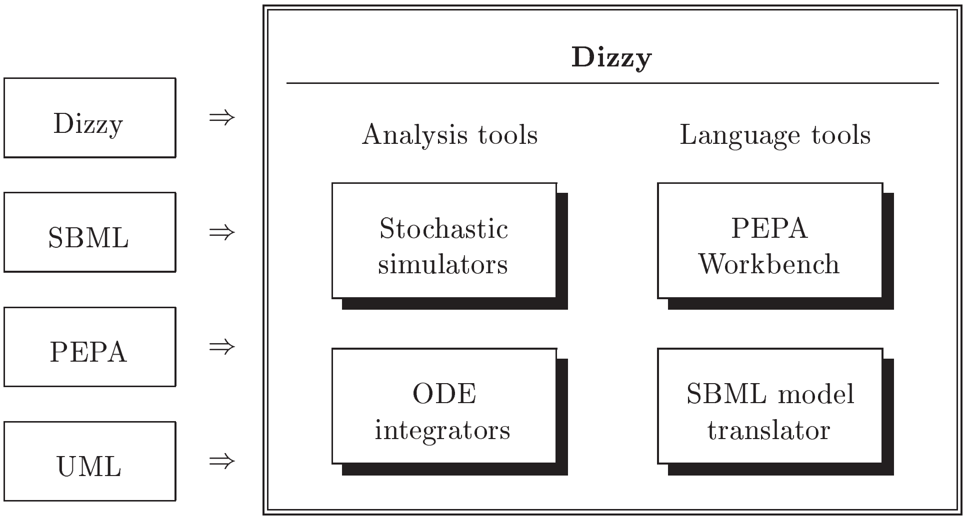


Fig. 10. Model processing on the Dizzy simulation and analysis platform

A screenshot of the Dizzy application processing the PEPA model of the secure electronic voting system is shown in Figure 11.

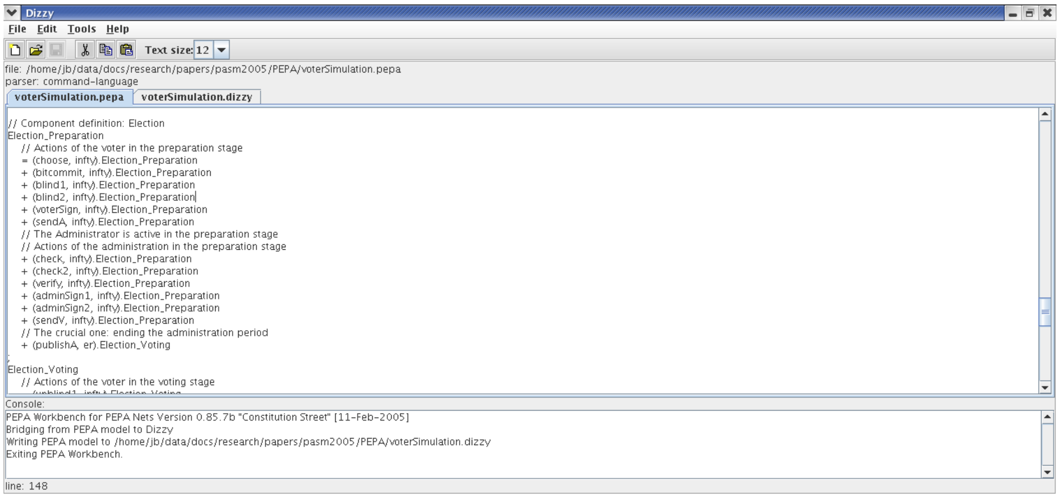


Fig. 11. Processing a PEPA model on the Dizzy platform.

Stochastic simulators for both Gillespie's Direct method [10] and the Gibson-Bruck method [11] are implemented in Dizzy. The Gibson-Bruck algorithm is $O(\log(M))$ in the number of reactions so it is preferred over the Gillespie algorithm for models with a large number of reactions and/or species. We found in practice for our model that Gillespie's method was faster, although we note that the availability of an implementation of Gibson-Bruck will be advantageous for more complex models.

A screenshot of the dialogue with respect to simulation parameters for Gillespie's Direct method is shown in Figure 12. (Recall that we use Gille-

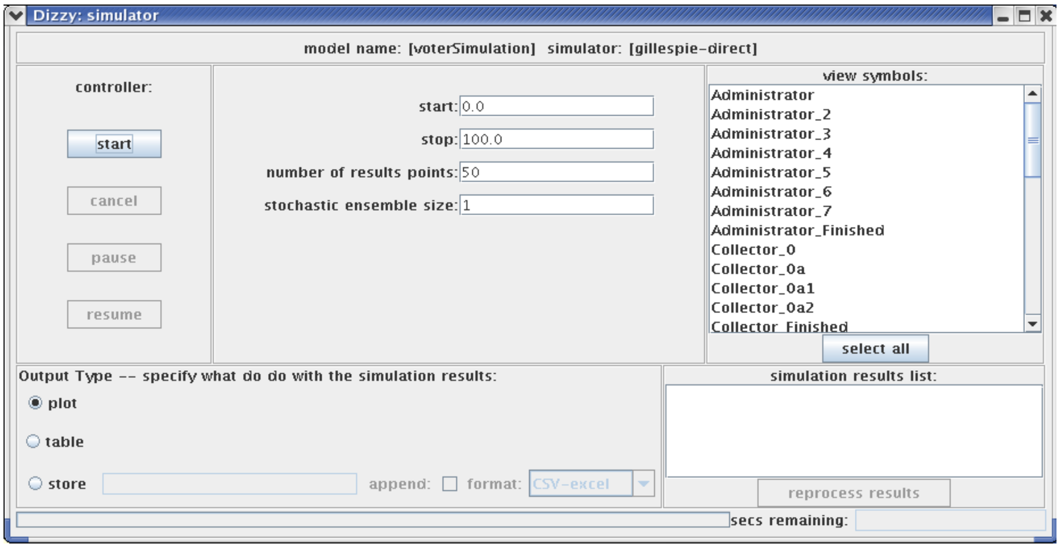


Fig. 12. Setting parameters for to simulate a PEPA model using Gillespie's Direct method. Simulation results from the Dizzy application can be rendered for on-line viewing or stored in a machine-processable format such as comma-separated values.

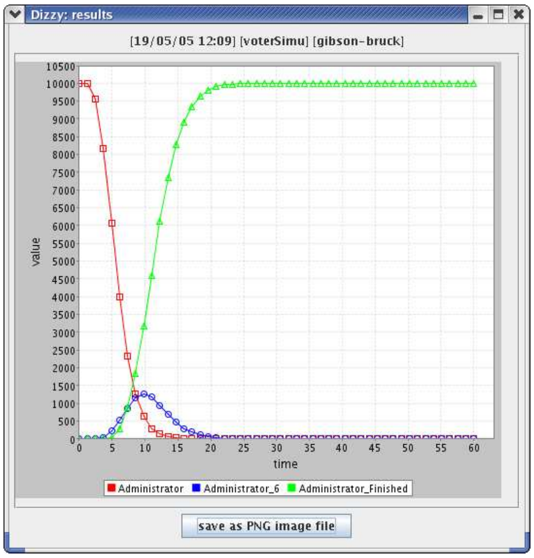


Fig. 13. Viewing simulation results from the Dizzy application interactively in a Java graphics window

spie's Direct method with rate functions which respect PEPA's apparent rate definition, not with the mass action semantics used in computational systems biology.)

When run, the implementation generates results as shown in Figure 13.

## 7 Conclusion

The aim of this paper is to show that stochastic simulation offers new possibilities for the analysis of massive state-space stochastic systems. By systematically converting a stochastic process algebra model to a set of chemical rate equations, we construct a continuum approximation of the number of components in a given state within that model. This gives us the advantage of being able to run a simulation of a process model which would be totally impractical using traditional discrete-event simulation techniques. To our knowledge, this is the first time that this style of simulation has been attempted within a stochastic process algebra framework.

To demonstrate the benefits of this method, we analysed a model of a secure electronic voting protocol with 10,000 voter agents. This system has approximately $10^{11750}$ states and is therefore not amenable to existing state-based analytic or simulation techniques. Further there is virtually no overhead in increasing the agent population size using this technique, rather the simulation performance is dependent on the log of the number of distinct rate equation generated by the model.

## Acknowledgements

In the course of producing stochastic simulation results for PEPA models, we had need to extend the open-source Java application, Dizzy, available from the Bolouri Group, Institute for Systems Biology. Stephen Ramsey from the Institute for Systems Biology, and principal architect of the Dizzy tool, was enormously helpful and responsive to our questions. Extensions to Dizzy to incorporate the PEPA Workbench were implemented by our students Adam Duguid and Erika Birse, and we thank them for doing this.

The PEPA Workbench is available from http://www.dcs.ed.ac.uk/pepa. The Imperial PEPA Compiler is available from http://www.doc.ic.ac.uk/ipc.

Jeremy Bradley is supported in part by the Nuffield Foundation under grant NAL/00805/G.

## References

[1] K. Zetter, "How e-voting threatens democracy." Wired Magazine article, Mar. 2004. http://www.wired.com/news/evote/0,2645,62790,00.html.

[2] "Machine politics." Wired Magazine feature on electronic voting, Apr. 2005. http://www.wired.com/news/evote/.

[3] L. C. Paulson, "The inductive approach to verifying cryptographic protocols," *J. Computer Security*, vol. 6, pp. 85–128, 1998.

[4] G. Lowe, "Breaking and fixing the Needham-Schroeder public-key protocol using FDR," *Software-Concepts and Tools*, vol. 17, no. 3, pp. 93–102, 1996.

[5] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. R. Nielson, "Control flow analysis can find new flaws too," in *Proceedings of Workshop on Issues in the Theory of Security (WITS 04)*, (Barcelona, Spain), Apr. 2004.

[6] N. Thomas, "Performability of a secure electronic voting algorithm," in *Proceedings of the first workshop on Practical Applications of Stochastic Modelling (PASM 2004)* (J. Bradley and W. Knottenbelt, eds.), (London, England), pp. 81–93, Sept. 2004. To appear in ENTCS.

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

[8] S. Gilmore, J. Hillston, and M. Ribaudo, "An efficient algorithm for aggregating PEPA models," *IEEE Transactions on Software Engineering*, vol. 27, pp. 449–464, May 2001.

[9] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM: Probabilistic symbolic model checker," in *Proceedings of the 12th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation* (T. Field, P. Harrison, J. Bradley, and U. Harder, eds.), vol. 2324 of *LNCS*, (London), pp. 200–204, Springer, 2002.

[10] D. Gillespie, "A general method for numerically simulating the stochastic time evolution of coupled chemical species," *J. Comp. Phys.*, vol. 22, pp. 403–434, 1976.

[11] M. Gibson and J. Bruck, "Efficient exact stochastic simulation of chemical systems with many species and many channels," *J. Comp. Phys.*, vol. 104, pp. 1876–1889, 2000.

[12] C. Priami, A. Regev, W. Silverman, and E. Shapiro, "Application of a stochastic name passing calculus to representation and simulation of molecular processes," *Information Processing Letters*, vol. 80, pp. 25–31, 2001.

[13] A. Phillips and L. Cardelli, "A correct abstract machine for the stochastic pi-calculus," in *Proceedings of BioConcur 2004*, 2004. To appear in Transactions on Computational Systems Biology.

[14] G. Clark, S. Gilmore, J. Hillston, and N. Thomas, "Experiences with the PEPA performance modelling tools," *IEE Proceedings—Software*, vol. 146, pp. 11–19, Feb. 1999. Special issue of papers from the Fourteenth UK Performance Engineering Workshop.

[15] F. Stathopoulos, "Enhancing the PEPA Workbench with simulation and experimentation facilities," Master's thesis, School of Computer Science, The University of Edinburgh, Sept. 2001.

[16] G. Clark and W. Sanders, "Implementing a stochastic process algebra within the Möbius modeling framework," in de Alfaro and Gilmore [32], pp. 200–215.

[17] K. Powell, "Rapid prototyping of high-performance concurrent Java applications," Master's thesis, School of Computer Science, The University of Edinburgh, Sept. 2002.

[18] R. McNab and F. Howell, "Using Java for Discrete Event Simulation," in *Proceedings of the Twelfth UK Computer and Telecommunications Performance Engineering Workshop (UKPEW)* (R. Pooley and J. Hillston, eds.), (The University of Edinburgh), pp. 219–228, 1996.

[19] G. Clark, *Techniques for the Construction and Analysis of Algebraic Performance Models*. PhD thesis, The University of Edinburgh, 2000.

[20] A. Benoit, M. Cole, S. Gilmore, and J. Hillston, "Enhancing the effective utilisation of Grid clusters by exploiting on-line performability analysis," in *Proceedings of the First International Workshop on Grid Performability* (S. Jarvis and N. Thomas, eds.), IEEE Computer Society Press, May 2005.

[21] S. Gilmore and J. Hillston, "The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling," in *Proceedings of the Seventh International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, no. 794 in Lecture Notes in Computer Science, (Vienna), pp. 353–368, Springer-Verlag, May 1994.

[22] J. Bradley, N. Dingle, S. Gilmore, and W. Knottenbelt, "Derivation of passage-time densities in PEPA models using IPC: The Imperial PEPA Compiler," in *Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems* (G. Kotsis, ed.), (University of Central Florida), pp. 344–351, IEEE Computer Society Press, Oct. 2003.

[23] J. T. Bradley, S. T. Gilmore, and N. Thomas, "How synchronisation strategy approximation in PEPA implementations affects passage time performance results," in *Applying Formal Methods: Testing, Performance, and M/E-Commerce (EPEW 2004)* (M. Núñez *et al*, ed.), vol. 3236 of *LNCS*, pp. 128–142, Springer-Verlag, Oct. 2004.

[24] S. Ramsey, D. Orrell, and H. Bolouri, "Dizzy: stochastic simulation of large-scale genetic regulatory networks," *J. Bioinf. Comp. Biol.*, vol. 3, no. 2, pp. 415–436, 2005.

[25] N. Thomas and J. Bradley, "Terminating processes in PEPA," in *Proceedings of the Seventeenth UK Performance Engineering Workshop* (K. Djemame and M. Kara, eds.), (University of Leeds), pp. 143–154, July 2001.

[26] A. Fujioka, T. Okamoto, and K. Ohta, "A practical secret voting scheme for large scale elections," in *ASIACRYPT '92: Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques*, (London, UK), pp. 244–251, Springer-Verlag, 1993.

[27] S. Gilmore, J. Hillston, M. Ribaudo, and L. Kloul, "PEPA nets: A structured performance modelling formalism," *Performance Evaluation*, vol. 54, pp. 79–104, Oct. 2003.

[28] A. Argent-Katwala, J. Bradley, and N. Dingle, "Expressing performance requirements using regular expressions to specify stochastic probes over process

algebra models," in *Proceedings of the Fourth International Workshop on Software and Performance*, (Redwood Shores, California, USA), pp. 49–58, ACM Press, Jan. 2004.

[29] J. Hillston and L. Kloul, "An efficient Kronecker representation for PEPA models," in de Alfaro and Gilmore [32], pp. 120–135.

[30] M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, and H. Kitano *et al.*, "The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models," *Bioinformatics*, vol. 19, no. 4, 2003.

[31] C. Canevet, S. Gilmore, J. Hillston, M. Prowse, and P. Stevens, "Performance modelling with UML and stochastic process algebras," *IEE Proceedings: Computers and Digital Techniques*, vol. 150, pp. 107–120, Mar. 2003.

[32] L. de Alfaro and S. Gilmore, eds., *Proceedings of the first joint PAPM-PROBMIV Workshop*, vol. 2165 of *Lecture Notes in Computer Science*, (Aachen, Germany), Springer-Verlag, Sept. 2001.

# A PEPA

## *A.1 Standard syntax*

This appendix provides a brief introduction to PEPA in order to make the paper self-contained. It can safely be skipped by anyone who already knows the PEPA language. For a full explanation which complements the brief description presented here the reader is referred to [7].

**Prefix:** The basic mechanism for describing the behaviour of a system with a PEPA model is to give a component a designated first action using the prefix combinator, denoted by a full stop. For example, $(\alpha, r).S$ carries out activity $(\alpha, r)$, which has action type $\alpha$ and an exponentially distributed duration with parameter $r$, and it subsequently behaves as $S$.

**Choice:** The component $P + Q$ represents a system which may behave either as $P$ or as $Q$. The activities of both $P$ and $Q$ are enabled. The first activity to complete distinguishes one of them: the other is discarded. The system will behave as the derivative resulting from the evolution of the chosen component.

**Constant:** It is convenient to be able to assign names to patterns of behaviour associated with components. Constants are components whose meaning is given by a defining equation. The notation for this is $X \stackrel{\mathit{def}}{=} E$. The name $X$ is in scope in the expression on the right hand side meaning that, for example, $X \stackrel{\mathit{def}}{=} (\alpha, r).X$ performs $\alpha$ at rate $r$ forever.

**Hiding:** The possibility to abstract away some aspects of a component's behaviour is provided by the hiding operator, denoted $P/L$. Here, the set $L$ identifies those activities which are to be considered internal or private to the component and which will appear as the unknown type $\tau$.

**Cooperation:** We write $P \underset{L}{\bowtie} Q$ to denote cooperation between $P$ and $Q$ over $L$. The set which is used as the subscript to the cooperation symbol, the *cooperation set* $L$, determines those activities on which the *cooperands* are forced to synchronise. For action types not in $L$, the components proceed independently and concurrently with their enabled activities. We write $P \parallel Q$ as an abbreviation for $P \underset{L}{\bowtie} Q$ when $L$ is empty.

However, if a component enables an activity whose action type is in the cooperation set it will not be able to proceed with that activity until the other component also enables an activity of that type. The two components then proceed together to complete the *shared activity*. The rate of the shared activity may be altered to reflect the work carried out by both components to complete the activity (for details see [7]).

In some cases, when an activity is known to be carried out in cooperation with another component, a component may be *passive* with respect to that activity. This means that the rate of the activity is left unspecified (denoted $\top$) and is determined upon cooperation, by the rate of the activity in the other component. All passive actions must be synchronised in the final model.

### *A.2 Derived forms and additional syntax*

We now describe some additional *derived forms* ("syntactic sugar") for PEPA. These do not add any expressive power to the language or require any semantic rules in addition to those in [7]. We have seen one derived form already: $P_1 \parallel P_2$ is a derived form for $P_1 \underset{\emptyset}{\bowtie} P_2$.

Because we are interested in transient behaviour we use the deadlocked process *Stop* as defined in [25] to signal a component which performs no further actions. We consider this to be simply an abbreviation for a deadlocked process, as shown below.

$$\mathit{Stop} \stackrel{\mathit{def}}{=} \Big( \big((a, r).\mathit{Stop}\big) \underset{\{a,b\}}{\bowtie} \big((b, r).\mathit{Stop}\big) \Big) / \{\, a, b \,\}$$

Finally, because we will be working with large numbers of copies of components, we introduce another abbreviation: we write $P[n]$ to denote $n$ copies of component $P$ executing in parallel. For example,

$$P[5] \equiv (P \parallel P \parallel P \parallel P \parallel P)$$

and refer to such an abbreviation as an *array* of components.