Performance analysis of Stochastic Process Algebra models using Stochastic Simulation

Jeremy T. Bradley[1], Stephen T. Gilmore[2], Nigel Thomas[3]

[1]Dept. of Computing, Imperial College London
180 Queen's Gate, London SW7 2BZ, UK.
jb@doc.ic.ac.uk

[2]LFCS, The University of Edinburgh
Edinburgh EH9 3JZ, UK.
Stephen.Gilmore@ed.ac.uk

[3]School of Computing Science University of Newcastle-upon-Tyne,
Newcastle-upon-Tyne NE1 7RU, UK.
Nigel.Thomas@ncl.ac.uk

Abstract

We present a translation of a generic stochastic process algebra model into a form suitable for stochastic simulation. By systematically generating rate equations from a process description, we can use tools developed for chemical and biochemical reaction analysis to provide time-series output for models with state spaces of $O(10^{10000})$ *and beyond. We apply these techniques to a significant case study: that of a secure electronic voting protocol.*

1 Introduction

A fundamental problem with traditional analysis of stochastic process algebra models is that they rely on the explicit representation of the global state space. Whether attempting steady-state, transient or passage-time analysis of such models, the so-called *state space explosion problem* limits the practical size and complexity of any model being studied.

One possible way round this problem is to represent the discrete state space of a model by a continuous approximation [18, 2]. This entails reducing the state of a stochastic process algebra model to a canonical form and then using a set of coupled rate equations to model the number of components or agents that are in a particular state at a time, t.

In traditional discrete-state transient analysis of a stochastic model, we are able to assign a probability to being in a particular state at a given time, t. Here, we use a stochastic simulator to provide trace executions of the underlying rate equations and use multiple traces to provide confidence intervals for being in a given state.

In this paper, we present a formal transformation from the stochastic process algebra, PEPA [16], to an equivalent rate equation description. We use an electronic voting system case study to show that a state-space of many times the size of an explicit state-space technique can be analysed.

1.1 Background: Stochastic simulation

The foremost techniques in present-day use for simulating biological and biochemical reactions involving significant numbers of molecules interacting in a volume are derived from Gillespie's Stochastic Simulation Algorithm (SSA) [10]. The significant achievement in Gillespie's work is to ground the derivation of his algorithm in the theory of statistical thermodynamics. This leads to an *exact* procedure for numerically simulating the dynamic evolution of a chemically reacting system. The method is accurate even at low copy numbers of reactants, where the assumption of continuity and the use of the law of mass action used in ordinary differential equation (ODE)-based analysis breaks down. Further, the SSA method converges, as the number of reactants increases, to the solution computed by the ODEs [26] so that the methods are in agreement in the limit, but SSA gives a more realistic representation when some reactants are present only in low numbers.

Gillespie's exact algorithm models systems in which there are M possible reactions represented by the indexed family R_μ $(1 \leq \mu \leq M)$. It builds on a *reaction probability density function* $P(\tau, \mu \mid \mathbf{X})$ such that $P(\tau, \mu \mid \mathbf{X})\mathrm{d}\tau$ is the probability that given the state $\mathbf{X}$ at time t, the *next* reaction in the volume will occur in the infinitesimal time interval $(t+\tau, t+\tau+\mathrm{d}\tau)$ *and* be an R_μ reaction. Starting from an initial state, SSA randomly picks the time and type of the next reaction to occur, updates the global state to record the fact that this reaction has happened, and then repeats.

The primary drawback of this method is that it rests on only one reaction happening in the given interval and thus the time step for the next reaction may need to be very small (driving up the cost of the simulation). The method was improved upon by Gibson and Bruck [9]. The strong points of Gibson's approach are the use of only one random number (Gillespie uses two) and taking time proportional to the logarithm of number of reactions. An *approximate* acceleration procedure called "τ-leaping" was later developed by Gillespie and Petzold [11]. The "implicit τ-leaping" method [23] was developed to attack the orthogonal problem of *stiffness*, common in multi-scale modelling, where different time-scales are appropriate for reactions. Recent advances in the field include the development of *slow-scale SSA* which produces a dramatic speed-up relative to SSA by prioritising rare events [4].

An excellent recent survey paper on stochastic simulation methods and their relation to differential-equation based analysis of reaction kinetics is [26].

2 PEPA

PEPA [16] is a parsimonious stochastic process algebra that can describe compositional stochastic models. These models consist of components whose actions incorporate random exponential delays.

The syntax of a PEPA component, P, is represented by:

$$P \quad ::= \quad (a, \lambda).P \;\Big|\; P + P \;\Big|\; P \underset{S}{\bowtie} P \;\Big|\; P/L \;\Big|\; A \qquad (1)$$

Prefix, $(a, \lambda).P$: This represents a process which does an action, a, and then becomes a new process, P. The time taken to perform a is described by an exponentially distributed random variable with parameter λ. The rate parameter may also take the value $\top$, which makes the action passive in a cooperation (see below).

Choice, $P_1 + P_2$: A race is entered into between components P_1 and P_2. If P_1 evolves first then any behaviour of P_2 is discarded and vice-versa. This is often called *competitive choice*.

Hiding, P/L: Actions in the set L that emanate from the component P are rewritten as silent τ-actions (with the same appropriate delays). The actions in L can no longer be used in cooperation with other components.

Constant, X: 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, $X \stackrel{\mathit{def}}{=} (a, r).X$ performs a at rate r forever.

Cooperation, $P_1 \underset{S}{\bowtie} P_2$: P_1 and P_2 run in parallel and synchronise over the set of actions in the set S. If P_1 is to evolve with an action $a \in S$, then it must first wait for P_2 to reach a point where it is also capable of producing an a-action, and vice-versa. In a cooperation, the two components then jointly produce an a-action with a rate that reflects the slower of the two components, $R = \frac{r_1}{r_a(P_1)} \frac{r_2}{r_a(P_2)} \min(r_a(P_1), r_a(P_2))$. In a passive cooperation, where P_1, say, can evolve with an $(a, \top)$-transition, the joint a-action inherits its rate from the P_2 component alone.

When two processes cooperate, PEPA uses a notion of *bounded capacity* to cap the overall rate of cooperating actions. Specifically, if the action-type $a \in S$ then the total observed rate of a when it is enabled in $P_1 \underset{S}{\bowtie} P_2$ can never exceed $\min(r_a(P_1), r_a(P_2))$. The function $r_a(P)$ is known as the apparent rate function and can be defined as:

$$r_a(P) = \sum_{P \xrightarrow{(a, \lambda_i)}} \lambda_i \qquad (2)$$

where $\lambda_i \in \mathbb{R}^+ \cup \{n\top \mid n \in \mathbb{Q}, n > 0\}$, $n\top$ is shorthand for $n \times \top$. As described above, $\top$ represents a passive action rate that will inherit the rate of the coaction from the cooperating component. $\top$ requires the following arithmetic rules:

$$\begin{aligned}
m\top < n\top \quad &: \quad \text{for } m < n \text{ and } m, n \in \mathbb{Q} \\
r < n\top \quad &: \quad \text{for all } r \in \mathbb{R}, n \in \mathbb{Q} \\
m\top + n\top = (m+n)\top \quad &: \quad m, n \in \mathbb{Q} \\
\frac{m\top}{n\top} = \frac{m}{n} \quad &: \quad m, n \in \mathbb{Q}
\end{aligned}$$

Note that $(r + n\top)$ is undefined for all $r \in \mathbb{R}$ in PEPA therefore disallowing components which enable both active and passive actions in the same action type at the same time, e.g. $(a, \lambda).P + (a, \top).P'$.

3 Rate Equation Simulation Models

In this section, we introduce the concept of a rate equation simulation model by showing how a client/server model would be translated into a typical simulation language. A simulation of PEPA model using rate equations is applicable for models which display massive parallelism in one of more component, for example:

$$A \underset{L}{\bowtie} A \underset{L}{\bowtie} \cdots \underset{L}{\bowtie} A$$

for some component A where:

$$\begin{aligned} A &\stackrel{\mathit{def}}{=} (a, \lambda).A' \\ A' &\stackrel{\mathit{def}}{=} (b, \mu).A \end{aligned}$$

A rate equation simulation in this case would simulate the number of components within the cooperation that were in state A or state A' at time, t.

Below is the PEPA description of a simple multi-client and multi-server model that has the appropriate parallel structure which makes it amenable to this style of analysis:

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

The system equation for the model is:

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

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*, λ, 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.

3.1 Rate equations

We are aiming to generate a set of rate equations representing the PEPA model. These will be input into a rate equation simulation tool e.g. [22], which is traditionally used to simulate chemical and biological processes. Fortunately, the tool gives us flexibility to migrate from the mass action semantics of the standard chemical modelling paradigm to a semantics which better matches that of PEPA. This issue only affects the cooperative actions within a model, in the client/server case it affects the overall rate of the *compute* action.

The formal translation of a PEPA model to a rate equation is covered in Section 4. Meanwhile, an informal procedure for generating a rate equation 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 and 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 $n(\mathit{Client}_1)$ components in state Client_1 then the overall observed rate of action *delay* will be $n(\mathit{Client}_1)\mu$. Combining this source/target state extraction with the rate calculations, we get the equivalent rate equations of Fig. 1.

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

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:

$$\begin{aligned} \min(C\top, S\lambda) &= \begin{cases} S\lambda & : C > 0 \\ 0 & : C = 0 \end{cases} \\ &= \theta(C)S\lambda \end{aligned} \tag{3}$$

$$\mathit{Client} + \mathit{Server} \xrightarrow{\theta(n(\mathit{Client}))n(\mathit{Server})\lambda} \mathit{Client}_1 + \mathit{Server}_1$$

$$\mathit{Client}_1 \xrightarrow{n(\mathit{Client}_1)\mu} \mathit{Client}$$

$$\mathit{Server}_1 \xrightarrow{n(\mathit{Server}_1)\nu} \mathit{Server}$$

where $\theta(x) = 1$ if $x > 0$, else 0.

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

```
// 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

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 α and β respectively.

3.2 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 [12]. The resulting file is shown in Fig. 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 `[ ]`.

4 Rate equation translation semantics

In order to perform rate equation simulation of a PEPA model, we need to look at a PEPA model in a slightly different but complementary way. Previously, a static system equation would be specified and this would serve the dual purpose of defining both the way in which the system was composed and the start state of the system. Now that we are considering systems of potentially millions of cooperating components, it becomes more useful to consider an aggregate and time-varying form of the same PEPA model.

The situation is best demonstrated by the system below:

$$\underbrace{P \parallel P \parallel \cdots \parallel P}_{n}$$

For large n, this system could be represented more succinctly by a vector which describes the *number* of components in a given derivative state. That is to say, suppose P has 2 other derivative states, P' and P'', in its component description. A 3-tuple vector (v_1, v_2, v_3) could be used to represent there being v_1 components in state P, v_2 in state P' and v_3 in state P'' in the cooperation above. Clearly $v_1 + v_2 + v_3 = n$, the total number of components in the cooperation. This has the effect of reducing the state space representation to an aggregated form (described in [13]) which requires a vector representation of size $D(P)$, the number of derivatives of P, rather than one of size n, in the unaggregated form. We further make the variables v_i functions of t, to show how the system evolves over time and we get the *time-based system equation* described below.

The start state of the system, at $t = 0$, is derivable from the original static system equation.

4.1 Time-based System Equation

The concept of a time-based system equation for a PEPA model was introduced in [3] and builds upon the *numerical vector form* concept from Hillston [18] to include extra information about cooperation and abstraction sets. The state of a PEPA model at time t can be represented by $P(t)$, which has the grammar:

$$P(t) ::= \\ P(t) \underset{L}{\bowtie} P(t) \;\Big|\; P(t)/L \;\Big|\; (N(S,t), \ldots, N(S,t))_L$$

where S is the derivative state of a sequential component. The *derivatives tuple* $(N(S_1,t), \ldots, N(S_{N_s},t))_L$ is used to count the number of derivatives of S in the cooperation:

$$S \underset{L}{\bowtie} S \underset{L}{\bowtie} \cdots \underset{L}{\bowtie} S$$

The derivatives tuple uses the function $N(X,t)$ to represent the number of components that are in state X at time t within the environment expressed by the overall PEPA system formula, $P(t)$. There are $N_s = |D(S)|$ elements in this tuple to represent the total number of derivative states of component S.

4.2 Stoichiometry function

We define the stoichiometry function, $\mathcal{F}(\cdot)$, to act over a time-based system equation version of a PEPA model to give a stoichiometric rate equation version of the same model. This stoichiometric form of a model expresses the number of components that need to be present in a system for a particular action to evolve, and further specifies how many resulting components are generated after evolution and with what rate the evolution occurred.

$$\mathcal{F} : Sys_T \to \mathcal{D} \tag{4}$$

where Sys_T is the set of time-based system equations and D is the power set of DZ.

$$\begin{aligned} DZ &= \{(a, \sum \omega_i S_i, \sum \omega_i' S_i, r) \\ &\quad : \ a \in \mathcal{A}ct, \omega_i, \omega_i' \in \mathbb{N}_0, S_i \in \mathcal{S}, r \in \mathcal{R}\} \end{aligned}$$

$\mathcal{S}$ is the set of sequential components and derivatives in a model. $\mathcal{R}$ is the set of time dependent rates. A member of the above set $(a, \sum \omega_i S_i, \sum \omega_i' S_i, r)$ would represent a single rate equation:

$$a : \quad \omega_1 S_1 + \cdots + \omega_n S_n \xrightarrow{r} \omega_1' S_1 + \cdots + \omega_n' S_n$$

which means that one way for the system to evolve an a action requires ω_i copies of S_i and produces ω_i' copies of S_i, for $1 \leq i \leq n$, and does so at rate r. This would occur in competition with other possible a evolutions (literally as a competitive choice in PEPA terms).

We define $\mathcal{F}(\cdot)$, the stoichiometry function of a PEPA model, over each of the possible structures of its time-based system equation as follows:

Pairwise cooperation: $\mathcal{F}(P_1(t) \bowtie_L P_2(t)) = N_1 \cup N_2 \cup C$, where:

$$\begin{aligned} N_i &= \{(a, E_1, E_2, r) \\ &\quad : \ (a, E_1, E_2, r) \in \mathcal{F}(P_i(t)), a \notin L\} \\ &\quad \text{for } i = 1, 2 \\ C &= \{(a, E_1 + F_1, E_2 + F_2, \min(r, s)) \\ &\quad : \ (a, E_1, E_2, r) \in \mathcal{F}(P_1(t)), \\ &\qquad (a, F_1, F_2, s) \in \mathcal{F}(P_2(t)), a \in L\} \end{aligned}$$

Action hiding: $\mathcal{F}(P(t)\backslash L) = N \cup H$, where:

$$\begin{aligned} N &= \{(a, E_1, E_2, r) \\ &\quad : \ (a, E_1, E_2, r) \in \mathcal{F}(P(t)), a \notin L\} \\ H &= \{(\tau, E_1, E_2, r) \\ &\quad : \ (a, E_1, E_2, r) \in \mathcal{F}(P(t)), a \in L\} \end{aligned}$$

Group cooperation:
$\mathcal{F}(\,(N(S_1,t), \ldots, N(S_n,t))_L) = N \cup C$, where:

$$\begin{aligned} N &= \{(a, S_i, S_j, N(S_i,t)\, r) \\ &\quad : \ (a, S_i, S_j, r) \in \mathcal{F}(S_i), \\ &\qquad 1 \leq i \leq n, a \notin L\} \end{aligned}$$

$$\begin{aligned} C &= \{(a, \sum_{i\,:\,\omega_i > 0} \omega_i S_i, \sum_{i,j\,:\,\omega_i > 0, \sigma_{ij} > 0} \sigma_{ij} S_j, \ \Gamma(\vec{\omega}, \sigma)) \\ &\quad : \ \vec{\omega} \in \mathcal{L}(a), \sigma \in \mathcal{M}'(a, \vec{\omega}), a \in L\} \end{aligned}$$

$$\begin{aligned} \Gamma(\vec{\omega}, \sigma) &= \min_i \{\lambda_{ij} \ : \ (\cdot, S_i, S_j, \lambda_{ij}) \in \mathcal{F}(S_i), \\ &\qquad \omega_i > 0, \sigma_{ij} > 0\} \end{aligned}$$

$$\begin{aligned} \mathcal{L}(a) &= \{\vec{\omega} \ : \ \vec{\omega} \in \mathbb{N}_0^n, 0 \leq \omega_j \leq N, \\ &\qquad 1 \leq j \leq n, \sum_{i=1}^{n} \omega_i = \sum_{i\,:\,r_a(S_i) > 0} \omega_i = N\} \end{aligned}$$

$$\begin{aligned} \mathcal{M}'(a, \vec{\omega}) &= \{\sigma \ : \ \sigma_{ij} = \pi_{i_j}, \vec{\pi}_i \in \mathcal{M}(a, \omega_i, S_i), \\ &\qquad 1 \leq i \leq n\} \end{aligned}$$

$$\begin{aligned} \mathcal{M}(a, m, X) &= \{\vec{\pi} \ : \ \vec{\pi} \in \mathbb{N}_0^n, 0 \leq \pi_j \leq m, \\ &\qquad 1 \leq j \leq n, \sum_{i=1}^{n} \pi_i = \sum_{i\,:\,(a, X, S_i, \cdot) \in \mathcal{F}(X)} \pi_i = m\} \end{aligned}$$

Sequential component:
$$\mathcal{F}(S) = \{(a, S, S', r) \ : \ S \xrightarrow{(a,r)} S'\}$$

5 Case study: secure electronic voting

Voting is the foundation of the democratic process. Electronic voting has many potential attractions in providing (ideally) ease of use and a quick, reliable count. Making electronic voting secure has been an active topic of research for more than twenty years and many secure electronic voting schemes have been introduced since the inception of anonymous channels to separate voters and votes by Chaum [5]. The most publicly visible form of secure voting is the use of online systems for voting in political elections which has been introduced in several countries. This form of voting has several obvious requirements:

Only registered voters are allowed to vote; voters only vote once; it should not be possible to find out who voted, or how they voted

These factors mean that any voting scheme for use in this scenario has to provide adequate authentication, vote management and so-called *blinding* mechanisms, while operating over a potentially insecure communication medium. Fujioka *et al* [8], formalised these requirements as *completeness* (all votes counted correctly), *soundness* (a dishonest voter cannot disrupt the election), *privacy* (of votes), *unreuseability* (cannot vote twice), *eligibility* (to vote), *fairness* (of the vote), *verifiability* (of the result). In addition, Iversen [20] introduced the requirement of *receipt freeness*; many protocols issue receipts or tokens of some form to prove to the voter that the system behaved as it should. However, these receipts might be used by a dishonest voter to prove that they voted in a certain way, thus facilitating vote selling.

Many secure voting schemes rely in some way on encrypting data and even with fast processors encryption and decryption adds an overhead to data processing. However, the major overheads arise because of the additional communication that is required in order to ensure that the requirements of the secure vote are met. Secure voting schemes will generally use some form of anonymous channel, digital pseudonyms, blind signatures, trusted authorities and multiple key ciphers to separate the voter, the authority to vote, the vote itself and the counting of the vote. Clearly there is a substantial overhead in providing these measures and therefore the performance of such a system is of obvious practical interest.

5.1 A secure electronic voting algorithm

This case study considers a secure electronic voting scheme proposed by Fujoika *et al* [8] which has been implemented in at least two systems, SENSUS [7] and EVOX [15]. Unlike other electronic voting schemes, such as Prêt à Voter [6], the scheme has been extended in [21] to incorporate multiple administrative domains to address some of the scalability issues that arise with a centralised system.

The scheme consists of an arbitrary number of voters, one or more administrators to issue authority to vote, and a teller system to collect votes and to determine the result. An anonymous channel is used to communicate the vote between the voter and the collector/counter. The scheme is outlined below:

Preparation: For a voter i: Choose the voting strategy. Commit to the strategy using a bit commitment scheme c_i. Blind the committed ballot, b_i. Sign the blinded ballot sv_i. Send to the administrator the signed blinded ballot, the blinded ballot and unique voter ID, ID_i.

Administration: For an administrator: Receive message from voter i. Check right to vote for voter i. Check voter i has not voted already. Verify the signature; if valid sign the blinded ballot, sa_i. Send sa_i to voter i. When the administration period is over, publish a list containing every $\{ID_j, b_j, sv_j\}$.

Voting: For a voter i: Unblind sa_i to give the ballot signed by the Administrator, ba_i. Check signature. Send $\{c_i, ba_i\}$ to the Counter through an anonymous channel.

Collecting: For a teller: Receive message from voter i. Check Administrators signature on ba_i; if valid add $\{N, c_i, ba_i\}$ to a list, where N is a unique reference number. When the collecting period is over, publish a list containing every $\{N, c_i, ba_i\}$.

Opening: For a voter i: Checks that the vote appears on the list published by the Counter; if not appeal. Send the bit commitment key k_i to the Counter through an anonymous channel.

Counting: For a teller: Use k_i to retrieve the voting strategy. Check the strategy is valid. When all votes are counted, publish the final result.

It is clear from this description that voting according to this scheme has to follow a prescribed sequence of events. It is reasonable to assume that an election will consist of a great many voters, generally thousands or perhaps hundreds of thousands in any given administrative domain, and millions in the election as a whole. At any given time there will be many voters wishing to cast their votes electronically and so the system has to be able to respond to multiple simultaneous requests at every stage of the process without hindering the voter by introducing unreasonable delays. As such, an analysis of this scheme should be able to determine the scalability (with respect to voters) of a given configuration of administrators and tellers.

An election occurs over a fixed time frame, typically of the order of 12 hours, during which all votes must be cast and following which counting will occur. From a performance perspective we can therefore deduce that the time taken to count the votes can be treated as a separate optimisation problem from the earlier phases. Furthermore it is imperative that the administration phase does not cause a bottleneck which might delay voters to such an extent as they are unable to cast their vote or lose interest or trust in the system. Therefore the throughput of voters in the administration phase is of key practical interest.

5.2 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 [24].

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 [24] 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 [24] for meaningful steady-state computation).

In contrast to [24] 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* [17] and are instead only alternative models of the same system.

Preparation, voting and opening in the election

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.

$$
\begin{array}{rcl}
\mathit{Voter}0 & \stackrel{\mathit{def}}{=} & (\mathit{choose}, c_1).\mathit{Voter}0\_1 \\
\mathit{Voter}0\_1 & \stackrel{\mathit{def}}{=} & (\mathit{bitcommit}, b_1).\mathit{Voter}0\_2 \\
\mathit{Voter}0\_2 & \stackrel{\mathit{def}}{=} & (\mathit{blind}_1, b_2).\mathit{Voter}0\_3 \\
\mathit{Voter}0\_3 & \stackrel{\mathit{def}}{=} & (\mathit{blind}_2, b_3).\mathit{Voter}0\_4 \\
\mathit{Voter}0\_4 & \stackrel{\mathit{def}}{=} & (\mathit{voter\_sign}, s_1).\mathit{Voter}0\_5 \\
\mathit{Voter}0\_5 & \stackrel{\mathit{def}}{=} & (\mathit{sendA}, s_2).\mathit{Voter}0\_5b \\
\mathit{Voter}0\_5b & \stackrel{\mathit{def}}{=} & (\mathit{sendV}, \top).\mathit{Voter}1 \\
\mathit{Voter}1 & \stackrel{\mathit{def}}{=} & (\mathit{unblind}_1, u_1).\mathit{Voter}1\_1 \\
\mathit{Voter}1\_1 & \stackrel{\mathit{def}}{=} & (\mathit{unblind}_2, u_2).\mathit{Voter}1\_2 \\
\mathit{Voter}1\_2 & \stackrel{\mathit{def}}{=} & (\mathit{verify}_1, v_2).\mathit{Voter}1\_3 \\
\mathit{Voter}1\_3 & \stackrel{\mathit{def}}{=} & (\mathit{verify}_2, v_3).\mathit{Voter}1\_4 \\
\mathit{Voter}1\_4 & \stackrel{\mathit{def}}{=} & (\mathit{sendC}, s_6).\mathit{Voter}2 \\
\mathit{Voter}2 & \stackrel{\mathit{def}}{=} & (\mathit{checkFail}, p \times c_4).\mathit{Voter}3 \\
 & + & (\mathit{checkSucc}, (1-p) \times c_4).\mathit{Voter}2b \\
\mathit{Voter}2b & \stackrel{\mathit{def}}{=} & (\mathit{sendCo}, s_7).\mathit{Voter\_Fin} \\
\mathit{Voter}3 & \stackrel{\mathit{def}}{=} & (\mathit{appeal}, a_1).\mathit{Voter}2b \\
\mathit{Voter\_Fin} & \stackrel{\mathit{def}}{=} & \mathbf{Stop}
\end{array}
$$

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.

$$
\begin{array}{rcl}
\mathit{Admin} & \stackrel{\mathit{def}}{=} & (\mathit{sendA}, \top).\mathit{Admin\_2} \\
\mathit{Admin\_2} & \stackrel{\mathit{def}}{=} & (\mathit{check}_1, c_2).\mathit{Admin\_3} \\
\mathit{Admin\_3} & \stackrel{\mathit{def}}{=} & (\mathit{check}_2, c_3).\mathit{Admin\_4} \\
\mathit{Admin\_4} & \stackrel{\mathit{def}}{=} & (\mathit{verify}, v_1).\mathit{Admin\_5} \\
\mathit{Admin\_5} & \stackrel{\mathit{def}}{=} & (\mathit{admin\_sign}_1, s_3).\mathit{Admin\_6} \\
\mathit{Admin\_6} & \stackrel{\mathit{def}}{=} & (\mathit{admin\_sign}_2, s_4).\mathit{Admin\_7} \\
\mathit{Admin\_7} & \stackrel{\mathit{def}}{=} & (\mathit{sendV}, s_5).\mathit{Admin\_Fin} \\
\mathit{Admin\_Fin} & \stackrel{\mathit{def}}{=} & \mathbf{Stop}
\end{array}
$$

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.

$$
\begin{aligned}
Col\_0 &\stackrel{def}{=} (sendC, \top).Col\_0a \\
Col\_0a &\stackrel{def}{=} (collector\_verify_1, v_4).Col\_0a1 \\
Col\_0a1 &\stackrel{def}{=} (collector\_verify_2, v_5).Col\_0a2 \\
Col\_0a2 &\stackrel{def}{=} (add, a_2).Col\_Fin \\
Col\_Fin &\stackrel{def}{=} \mathbf{Stop}
\end{aligned}
$$

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.

$$
\begin{aligned}
Count\_1 &\stackrel{def}{=} (sendCo, \top).Count\_1a \\
Count\_1a &\stackrel{def}{=} (check\_strategy, c_5).Count\_Fin \\
Count\_Fin &\stackrel{def}{=} \mathbf{Stop}
\end{aligned}
$$

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 [14] and with the *stochastic probes* [1] 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* [19]. 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 θ function would be a very suitable way in general to implement functional rates.

$$
\begin{aligned}
Elect\_Prep &\stackrel{def}{=} (choose, \top).Elect\_Prep \\
&+ (bitcommit, \top).Elect\_Prep \\
&+ (blind_1, \top).Elect\_Prep \\
&+ (blind_2, \top).Elect\_Prep \\
&+ (voter\_sign, \top).Elect\_Prep \\
&+ (sendA, \top).Elect\_Prep \\
&+ (check_1, \top).Elect\_Prep \\
&+ (check_2, \top).Elect\_Prep \\
&+ (verify, \top).Elect\_Prep \\
&+ (admin\_sign_1, \top).Elect\_Prep \\
&+ (admin\_sign_2, \top).Elect\_Prep \\
&+ (sendV, \top).Elect\_Prep \\
&+ (publishA, er).Elect\_Voting \\
Elect\_Voting &\stackrel{def}{=} (unblind_1, \top).Elect\_Voting \\
&+ (unblind_2, \top).Elect\_Voting \\
&+ (verify_1, \top).Elect\_Voting \\
&+ (verify_2, \top).Elect\_Voting \\
&+ (sendC, \top).Elect\_Voting \\
&+ (collector\_verify_1, \top).Elect\_Voting \\
&+ (collector\_verify_2, \top).Elect\_Voting \\
&+ (add, \top).Elect\_Voting \\
&+ (publishC, er).Elect\_Count \\
Elect\_Count &\stackrel{def}{=} (checkFail, \top).Elect\_Count \\
&+ (checkSucc, \top).Elect\_Count \\
&+ (sendCo, \top).Elect\_Count \\
&+ (appeal, \top).Elect\_Count \\
&+ (check\_strategy, \top).Elect\_Count \\
&+ (final\_publish, er).Elect\_Fin \\
Elect\_Fin &\stackrel{def}{=} \mathbf{Stop}
\end{aligned}
$$

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

$$
Elect\_Prep \underset{\mathcal{L}}{\bowtie} Electoral\_Personae
$$

$$
\begin{aligned}
Electoral\_Personae &\stackrel{def}{=} Voter_0[N] \underset{\mathcal{M}}{\bowtie} Electoral\_App \\
Electoral\_App &\stackrel{def}{=} Col_0[N] \mid\mid Count_1[N] \mid\mid Admin[N]
\end{aligned}
$$

and:

$$
\begin{aligned}
N &= 10,000 \\
\mathcal{L} &= \{choose, bitcommit, blind_1, blind_2, voter\_sign, \\
&\quad sendA, sendV, unblind_1, unblind_2, verify_1, verify_2, \\
&\quad sendC, checkFail, checkSucc, sendCo, appeal, \\
&\quad publishA, check_1, check_2, verify, admin\_sign_1, \\
&\quad admin\_sign_2, collector\_verify_1, collector\_verify_2, \\
&\quad add, publishC, check\_strategy, final\_publish\} \\
\mathcal{M} &= \{sendA, sendV, sendC, sendCo, publishC\}
\end{aligned}
$$

5.3 Results

The models presented above are now converted to rate equations using the techniques of Section 4, then analysed numerically using data derived from an implementation of the voting scheme. The data is based on using RSA with a key length of 1024 bits, a maximum bit commitment length of 50 bits, a random padding of 100 bytes per message and a mix message block size of 110 bytes. By far the most significant time delays in the scheme are the decryption of the blinded votes and revelation messages. Other significant delays are encountered in the communication involved in sending the various messages and the overhead in signing the blinded messages. All other actions are very fast by comparison. This has the effect of making the resultant underlying continuous time Markov chain very stiff. Experiments with the implementation showed that the system is particularly sensitive to the padding length and mix message block lengths as these impact the slowest operations.

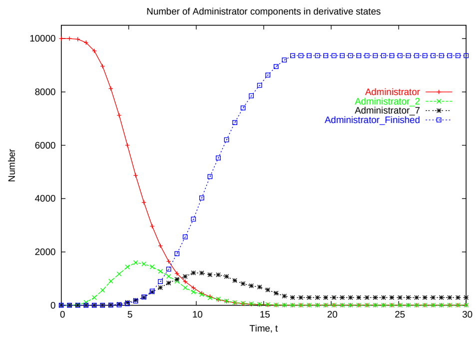

Fig. 3. A simulation of the Administrator component

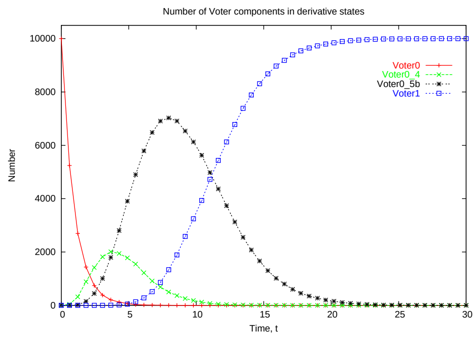

Fig. 4. A simulation of the Voter component through its early evolution to $Voter1$

Figs. 3 to 5 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.

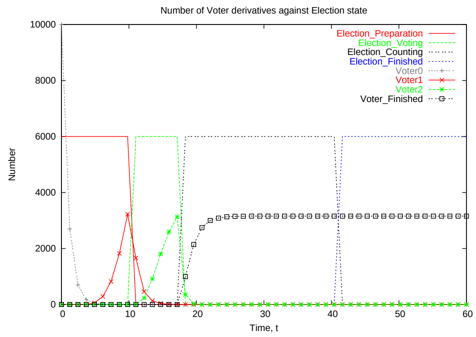

Fig. 5. A joint simulation of $Voter$ and $Elect$ components where the phases of the $Voter$ follow those of the $Elect$

In Fig. 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 $Admin_2$ and $Admin_7$ are transient states of the component and so the populations here almost approach 0. The last state and also the absorbing state of the component is $Admin\_Fin$, which ends up with the bulk of the population in this trace.

The simulations of the Voter component are shown in Figs. 4. It shows the smooth evolution of $Voter$ to derivative $Voter_1$. There is a close relationship between $Elect$ and $Voter$ that can be seen more closely in Fig. 5 and involves the later stages of the $Voter$ lifecycle.

Fig. 5 shows the inherent synchronisation between $Voter$ and $Elect$ 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 $Elect$ component, in its state change to $Elect\_Voting$ and $Elect\_Count$ respectively. The end of the final Election phase is not seen by the Voter as it concerns the completion of counting the votes.

6 Conclusion

In this paper, we have significantly extended the contribution of [2] by providing a formal translation of a generic PEPA model into a stochastic simulation model. Using the stoichiometric function $\mathcal{F}(\cdot)$ and a novel representation of a PEPA system equation from [3], we can generate rate equations for the PEPA models which can be simulated using tools such as Dizzy.

With these new techniques, we have carried out simulations on a significant case study of electronic voting protocol from [24, 2]. The representation of the voting system as a simulation has enabled us to analyse a state space of $O(10^{10000})$ states; far beyond the capability of traditional explicit state-space representation techniques.

References

[1] 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*, pages 49–58, Redwood Shores, California, USA, Jan. 2004. ACM Press.

[2] J. T. Bradley and S. T. Gilmore. Stochastic simulation methods applied to a secure electronic voting model. In N. Thomas, J. T. Bradley, and W. J. Knottenbelt, editors, *PASM'05, Proceedings of 2nd International Workshop on Practical Applications of Stochastic Modelling*, Newcastle, July 2005.

[3] J. T. Bradley, S. T. Gilmore, and J. Hillston. Using continuous state-space approximation of process algebra models to analyse internet worm attacks. In *SIGMETRICS'06, Proceedings of Joint International Conference on Measurement and Modeling of Computer Systems*, St. Malo, July 2006. ACM. (submitted).

[4] Y. Cao, D. Gillespie, and L. Petzold. Accelerated stochastic simulation of the stiff enzyme-substrate reaction. *Journal of Chemical Physics*, 123:144917–1 – 144917–12, 2005.

[5] D. Chaum. Untraceable electronic mail, return addresses and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

[6] D. Chaum, P. Ryan, and S. Schneider. A practical, voter-verifiable election scheme. Tech. Rep. CS-TR 880, School of Computer Science, University of Newcastle, December 2004.

[7] L. Cranor and R. Cryton. Sensus: A security-conscious electronic polling system for the internet. In *Proceedings of the Thirtieth Hawaii International Conference on System Sciences (HICSS 30)*, pages 561–570. IEEE Computer Society, 1997.

[8] 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*, pages 244–251, London, 1993. Springer-Verlag.

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

[10] D. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81(25):2340–2361, 1977.

[11] D. Gillespie and L. Petzold. Improved leap-size selection for accelerated stochastic simulation. *J. Comp. Phys.*, 119:8229–8234, 2003.

[12] S. Gilmore and J. Hillston. The PEPA workbench: A tool to support a process algebra-based approach to performance modelling. In G. Haring and G. Kotsis, editors, *Proceedings of the 7th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, volume 794 of *Lecture Notes in Computer Science*, pages 353–368. Springer-Verlag, Vienna, May 1994.

[13] S. Gilmore, J. Hillston, and M. Ribaudo. An efficient algorithm for aggregating PEPA models. *IEEE Transactions on Software Engineering*, 27(5):449–464, May 2001.

[14] S. Gilmore, J. Hillston, M. Ribaudo, and L. Kloul. PEPA nets: A structured performance modelling formalism. *Performance Evaluation*, 54(2):79–104, Oct. 2003.

[15] M. Herschberg. Secure electronic voting over the world wide web. MEng thesis, MIT, 1997. `http://theory.lcs.mit.edu/$^\sim$cis/voting/voting.html`.

[16] J. Hillston. *A Compositional Approach to Performance Modelling*, volume 12 of *Distinguished Dissertations in Computer Science*. Cambridge University Press, 1996. ISBN 0 521 57189 8.

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

[18] J. Hillston. Fluid flow approximation of PEPA models. In *QEST'05, Proceedings of the 2nd International Conference on the Quantitative Evaluation of Systems*, Italy, September 2005. IEEE Computer Society Press.

[19] J. Hillston and L. Kloul. An efficient Kronecker representation for PEPA models. In L. de Alfaro and S. Gilmore, editors, *Proceedings of the first joint PAPM-PROBMIV Workshop*, volume 2165 of *Lecture Notes in Computer Science*, pages 120–135, Aachen, Germany, Sept. 2001. Springer-Verlag.

[20] K. Iversen. A cryptographic scheme for computerised general elections. In J. Feigenbaum, editor, *Advances in Cryptology – Proceedings of CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 405–419. Springer-Verlag, 1991.

[21] R. Joaquim, A. Zuquete, and P. Ferreira. Revs a robust electronic voting system. In *Proceedings of IADIS International Conference e-Society 2003*, volume 1, pages 95–103, 2003.

[22] S. Ramsey, D. Orrell, and H. Bolouri. Dizzy: stochastic simulation of large-scale genetic regulatory networks. *J. Bioinf. Comp. Biol.*, 3(2):415–436, 2005.

[23] M. Rathinam, L. Petzold, Y. Cao, and D. Gillespie. Stiffness in stochastic chemically reacting systems: The implicit tau-leaping methodb. *Journal of Chemical Physics*, 119(24):12784–12794, Dec. 2003.

[24] N. Thomas. Performability of a secure electronic voting algorithm. In J. T. Bradley and W. J. Knottenbelt, editors, *PASM'04, Practical Applications of Stochastic Modelling*, pages 81–94, The Royal Society, London, September 2004. Imperial College London.

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

[26] T. Turner, S. Schnell, and K. Burrage. Stochastic approaches for modelling in vivo reactions. *Computational Biology and Chemistry*, 28:165–178, 2004.