Analysing Performance of Lift Systems in PEPA

Amani El-Rayes and Marta Kwiatkowska and Steven Minton*

Abstract

We use the stochastic process algebra PEPA [8] to specify lift systems and analyse their performance. We focus on the mean passenger waiting time versus the speed of the lift (given by the slowest rate such as the closing of the doors) and the rate of passengers arrivals. The results are obtained by compiling the specification in the PEPA workbench, and then solving the resulting equilibrium state equation in Matlab. The outcome of the PEPA analysis is compared with traditional engineering methods for lift traffic analysis known from the literature [2]. We find that PEPA has potential for finer grain analysis than existing methods. Finally, we briefly discuss our experience with the PEPA workbench.

1 Introduction

Performance modelling is concerned with the capture of the dynamic behaviour of computer and communication systems, and with their subsequent analysis. Traditionally, analytical and numerical techniques such as Markov chain analysis are applied, as in e.g. queuing theory, but they suffer from the following problems: for complex systems the models become large and unwieldy, and the structure of the solution rarely corresponds to the often hierarchical structure of the system. A relatively recent proposal is to combine *process algebras*, i.e. specification languages such as CCS [12] which allow compositional design, with stochastic modelling, usually in terms of Markov processes. This is typically achieved by augmenting the process algebra notation, where actions are assumed to be instantaneous, with timing information, usually an exponentially distributed random variable. As examples

*School of Computer Science, University of Birmingham, Edgbaston, Birmingham B15 2TT, UK. Email: {ahe,mzk,smm}@cs.bham.ac.uk

of thus obtained *stochastic process algebras* we mention PEPA (Performance Evaluation Process Algebra) [8], TIPP [7] and EMPA [3].

We use PEPA, and more specifically the PEPA workbench [4], to model and analyse a variety of lift systems. Although the PEPA tool is still in early stages of development and rather rudimentary, it has already been successfully applied to industrial problems, e.g. the robot control system [5, 10]. However, all the models encountered by us were *linear* in structure, as is often the case with production lines. Our aim is to evaluate PEPA from the point of view of modelling and analysing a *highly concurrent* and *non-deterministic* system. We choose lift systems as an object of our attention for the following reasons:

- they range from simple to quite complex multi-lift systems, with a high degree of parallelism and non-determinism present (e.g, if we consider a lift serving n floors then at any one time it may receive n calls from the floors and a number of requests to go to one of the n floors)
- their performance, e.g. mean waiting time, is sensitive to the timing information, type of distribution, as well as the actual control algorithm, yielding a non-trivial domain for a stochastic analysis
- established traffic analysis methododology exists for lift systems, see e.g. [2], thus making comparisons possible.

As we mentioned before, the PEPA workbench is still under development, and, in particular, handling models with large number of states in the tool is rather tedious. To cater for this, we shall aim for our specifications to be *scalable*, that is, specified and analysed for a small number of floors, which can then be transformed in a straightforward manner (when feasible for the workbench) to a full-scale model, and also as *accurate* as possible. When analysing performance we focus on the mean waiting time characteristics, but other performance measures can also be considered.

Some techniques have been presented for model simplification and state space aggregation to make it easier to tackle large problems; these methods typically require generation of the original state space before they can be applied. We reject those in favour of detailed description as we believe that simplifications lead to approximate characteristics, which conflicts with one of our goals. Techniques also have been proposed that operate at the level of the PEPA model, thus avoiding the generation of the original state space, see e.g. [9]. Whilst undoubtedly useful, these methods as they currently stand are unlikely to have a major bearing the main arguments put forward in this paper.

The outcome of our evaluation should provide useful feedback for future development of the PEPA tool.

2 The PEPA language and workbench

In this section we overview the PEPA language [8] and the workbench [5].

In performance modelling, an appropriate representation of a system is used to capture the essential characteristics of that system, so that its performance can be reproduced. Such models are usually based on stochastic models, from which performance measures can be easily obtained.

PEPA is intended to maintain many of the characteristics of a process algebra, whilst incorporating the necessary features to make it suitable for specifying stochastic processes. In particular, the stochastic processes underlying PEPA models are continuous time Markov processes, which can be solved for a steady state probability distribution.

2.1 Syntax and semantics of PEPA

In PEPA, a system is modelled as a set of *components* which perform *actions*, either individually or multiply. Each component will correspond to some element of the system which can be considered in isolation, be it a physical component or a mode of behaviour. The actions correspond to the possible activities of that element.

The actions themselves have associated with them a *rate*. This determines how quickly that activity may occur, and distinguishes PEPA from other process algebras where such actions are instantaneous. Each activity consists of a pair, (α, r), of the activity name and its associated rate. The rate will be either a parameter of an exponential distribution (so that if the rate is λ, the expected time at which the action occurs is $1/\lambda$ with the probability $P(t_0 < t) = 1 - e^{-\lambda t}$), or the distinguished symbol $\top$, which can be read as *unspecified*. In this latter case, the action is performed in communication with another component, and this component is passive, in that it may only perform the action when the other component does.

The exclusive use of the exponential distribution is a necessary feature of PEPA as it stands. In particular, specifying activities with constant duration, say t_0, or complying with Poisson or other distributions is not possible at present. The restriction to exponential distributions is needed to guarantee that the underlying stochastic process is a continuous time Markov process.

Components and activities can be combined in a number of ways. The following constructions allow for components to be expressed in terms of other components, and give PEPA its expressive power.

The syntax for terms in PEPA is as follows:

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

where $(\alpha, r).P$ denotes prefix, $P{+}Q$ choice, $P \underset{L}{\bowtie} Q$ cooperation, P/L hiding, X variable and $A \stackrel{def}{=} P$ constant.

The component $(\alpha, r).P$ commences by performing activity α, which takes some time δt drawn from the distribution, and then behaves as component P. $P + Q$ may perform as either P or Q, which is possible even if they begin with the same activity. $P \underset{L}{\bowtie} Q$ consists of the two components, P and Q, proceeding independently, except that they cooperate over all action types represented in the set L. In other words, each component can perform any activities not found in the set L entirely independently. However, they may only commence performing an activity from L (such activities are called *shared*) when they are both in a position to do so. The component P/L acts as component P, except that activities whose type occurs in L are *hidden*, meaning that their type is not witnessed upon completion. Instead, they appear as the unknown type τ, and can be regarded as an internal delay by the component. Constants $A \stackrel{def}{=} P$ are components whose meaning is given by a defining equation such as $A \stackrel{def}{=} P$ which gives the constant A the behaviour of the component P.

For a PEPA model we can derive a derivation graph (a multigraph in which terms are nodes and arcs represent the transitions between them). The stochastic process underlying the PEPA model has as *states* the nodes of the graph, and the *transition rate* between states is the sum of the rates labelling arcs between the corresponding nodes. Formally, the stochastic process $X(t)$ determined by a PEPA model is the Markov process $X(t) = C_i$, meaning that the system behaves as the component C_i at time t. The infinitesimal generator matrix Q of the Markov process is formed by taking the transition rate $Q(C_i, C_j)$ as the off-diagonal elements Q_{ij}, and the negative sum $-\Sigma_{j \neq} Q_{ij}$ as the diagonal elements Q_{ii}. Under certain conditions (i.e. if the Markov process is time-homogenous irreducible whose states are positive recurrent, and also strongly connected and finite-state) the equilibrium probability distribution Π can be computed by solving the equation $\Pi = Q0$ (subject to the normalising condition $\Sigma_i \Pi(C_i) = 1$).

2.2 Performance measures in PEPA

In order to derive a performance measure in PEPA *reward structures* are used. PEPA differs from the traditional Markov process modelling by being *action based*, as opposed to *state based*, and so rewards can be associated with activities. The performance measure is defined as the total reward R based on the steady state probability distribution Π of the underlying Markov process, i.e. $R = \Sigma_i \rho_i \Pi(C_i)$.

A typical state-based measure is utilisation; to calculate this associate a reward of 1 with the states in which a resource is used, and 0 otherwise, and then calculate the total reward. We often require action-based measures, i.e. those in which the rate of an activity must be taken into account. An example of such is the average rate of arrival, which is obtained by taking the arrival rate as the reward and multiplying it with the probability of being in one of the states from which the activity may occur. Combinations of state- and action-based measures, such as the mean waiting time, are also used.

2.3 The PEPA workbench

The PEPA workbench inputs the model (.pepa text file) and after checking for syntax errors, it computes the transition rate matrix Q (.m file). This file, intended for use with a suitable mathematics package (in our case Matlab), can be used to solve the underlying Markov process for the steady state probability distribution Π necessary for performance analysis. As PEPA is in an early stage of developemnt, no tools are provided for automatic or semi-automatic calculation of desired performance measures directly from the symbolic representation (i.e. directly from the PEPA model). Instead, such calculations are performed at the level of Matlab, which requires translation between the symbolic representation of states and the corresponding indices in the transition rate matrix (two more files, .table and .hash, are generated to ease the translation process). Understandably, the handling of models with large numbers of states is rather tedious, although the workbench and the Matlab package have been known to solve the steady state equation for models of over 100,000 of states.

3 A simple lift system

We first illustrate our approach by analysing a simple lift system in PEPA.

Consider a simple one-person lift operating between two floors, which is served on each floor by a one-person queue. It must wait on a floor until someone arrives from the associated queue, or until there is a call from a person on the other floor. Then it must move to the other floor, and subsequently behave as before. These two modes of behaviour correspond to the states $Lift_0$ and $Lift_1$ when the lift is on the 'ground' floor, and to states $Lift_2$ and $Lift_3$ when the lift is on the first floor.

The two queues are independent, but communicate with the lift via the activities $ArrivalN$, $CallN$ and $NoArrivalN$ where $N = 0, 1$. PEPA does not offer parametrization, and so the floor index N has to be instead encoded in the process identifier. When the lift first arrives at floor N, it can only accept an arrival from the queue, or the message $NoArrivalN$. This is to make sure that the lift does not wait on floor N if there is no-one waiting there, in which case the lift is ready to accept a call from the other floor.

$$
\begin{aligned}
Lift_0 &\stackrel{def}{=} (Arrival0, \top).Lift_1 + (NoArrival0, \top).Lift_{0a} \\
Lift_{0a} &\stackrel{def}{=} (Arrival0, \top).Lift_1 + (Call1, \top).Lift_1 \\
Lift_1 &\stackrel{def}{=} (Up, r_1).Lift_2 \\
Lift_2 &\stackrel{def}{=} (Arrival1, \top).Lift_3 + (NoArrival1, \top).Lift_{2a} \\
Lift_{2a} &\stackrel{def}{=} (Arrival1, \top).Lift_3 + (Call0, \top).Lift_3 \\
Lift_3 &\stackrel{def}{=} (Down, r_2).Lift_0
\end{aligned}
$$

$$
\begin{aligned}
Queue0_0 &\stackrel{def}{=} (OneOn0, r_3).Queue0_1 + (NoArrival0, r_4).Queue0_0 \\
Queue0_1 &\stackrel{def}{=} (Arrival0, r_4).Queue0_0 + (Call0, r_5).Queue0_1
\end{aligned}
$$

$$
\begin{aligned}
Queue1_0 &\stackrel{def}{=} (OneOn1, r_6).Queue1_1 + (NoArrival1, r_4).Queue1_0 \\
Queue1_1 &\stackrel{def}{=} (Arrival1, r_7).Queue1_0 + (Call1, r_8).Queue1_1
\end{aligned}
$$

$$
(Queue0_0 \underset{\emptyset}{\bowtie} Queue1_0) \underset{\mathcal{S}}{\bowtie} Lift_0
$$
$$
\text{where } \mathcal{S} = (Arrival0, Arrival1, Call0, Call1)
$$

Suppose the lift is on the ground floor, and there is no new arrival on ground floor. A possible scenario in this lift is: a person joins the first-floor queue (via $OneOn1$), calls the lift ($Call1$), the lift ascends (Up), the passenger enters the lift ($Arrival1$), and the lift returns ($Down$).

For the analysis, we concentrate on the mean waiting time for passengers on the ground floor. We use Little's Law, which states that the average number of entities in a system is equal to the product of the average rate at which entities arrive and the average time an entity is resident in the system. Thus, we need to find the average number of people waiting, and the average rate at which they arrive. The average number of people waiting is obtained

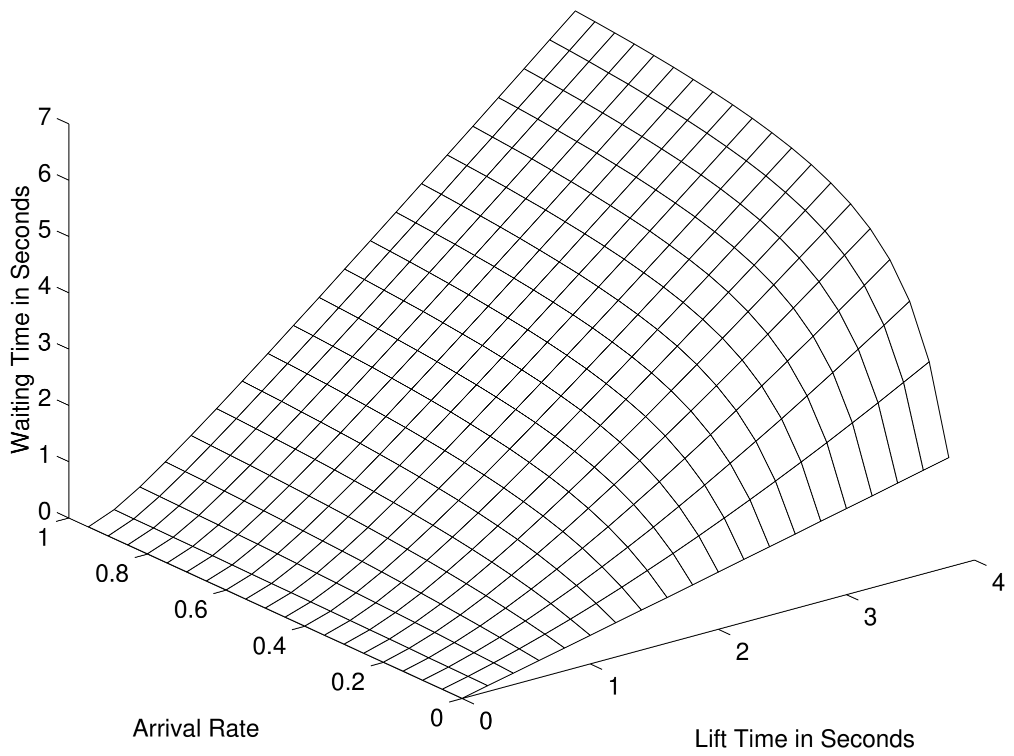

Figure 1: Performance of a Simple Lift

by associating a reward 1 (in general, the number of people waiting, but in the simplified model we assumed 1-person queues) with all states in which an arrival activity (*Arrival*0) is enabled. The average rate at which people arrive is given by associating the rate of arrival with the arrival activity, and multiplying it by the probability of being in one of the states from which arrival can occur (i.e. the states corresponding to the queue being empty). This means associating a reward of r_3 with the activity *OneOn*0.

We summarise the results in a three-dimensional graph of average waiting time against a measure of lift speed (the average time taken for the *Up* and *Down* actions, r_1 and r_2) and arrival rate (the values r_3 for ground floor and r_6 for first floor). Since the mean of an exponential distribution with parameter μ is $1/\mu$, the lift time is given by the reciprocal of the rate r_1 ($= r_2$). The remaining rates are assumed to be insignificant ($= 500$). The underlying stochastic model has 24 states.

This exhibits the kind of behaviour we might expect. For low lift times (in other words for a fast lift) the waiting time is low, even for high arrival rates. On the other hand, if the lift is slow, the waiting time quickly builds up as the arrival rate increases, resulting in a disproportionate 'jump'.

The PEPA workbench does not allow any execution traces of the model, and so it is possible for errors in the model to remain undetected for some time. We further gain confidence in our model by the following analysis. An arrival rate of one corresponds, on average, to one arrival every second. Consider this rate coupled with a lift time of four seconds. Now suppose the lift is on the ground floor. It departs for the first floor, and, one second after it leaves, there is an arrival on the ground floor. Three seconds later, the lift arrives at the first floor, and it will take a further four seconds to return to the ground floor. Thus, the arrival must wait seven seconds, which is consistent with the graph. Notice that if we did not have the activity $NoArrivalN$ then the waiting time could double, as demonstrated in the following scenario. Suppose that there are arrivals at each floor before the lift arrives at the first floor. When the lift arrives, instead of, as expected, the person on the first floor getting in, the lift could respond to the call on the ground floor, and immediately return there, thus forcing the passenger to wait longer.

We are now in a position to exploit the compositionality of PEPA and quickly create an interesting variation on our lift, by extending the model to include more than one lift. We need only alter the last line:

$$(Queue0_0 \underset{\emptyset}{\bowtie} Queue1_0) \underset{\mathcal{S}}{\bowtie} (Lift_0 \underset{\emptyset}{\bowtie} Lift_0)$$
$$\text{where } \mathcal{S} = (Arrival0, Arrival1, Call0, Call1)$$

The analysis yields the graph given in Figure 2. The underlying stochastic process has 144 states, as opposed to 24 for the previous model. We can see that this system has similar characteristics to the previous one, but is approximately twice as fast.

We should point out that, in general, such a simplistic approach to the increase in complexity would not work, as we must ensure that there is no possible timing of arrivals and lifts that leads to a deadlock situation, which is not always easily seen by considering a single component – in fact, 'multiplying' components may result in the introduction of deadlock where deadlock was not originally present. In the case of deadlock, the solution to the steady state equation, if it can be approximated, yields probability 1 for the deadlocked state and 0 for remaining states. The workbench has no support for automatic deadlock detection at present.

The above description can now serve as a basis for a variety of performance and cost measures.

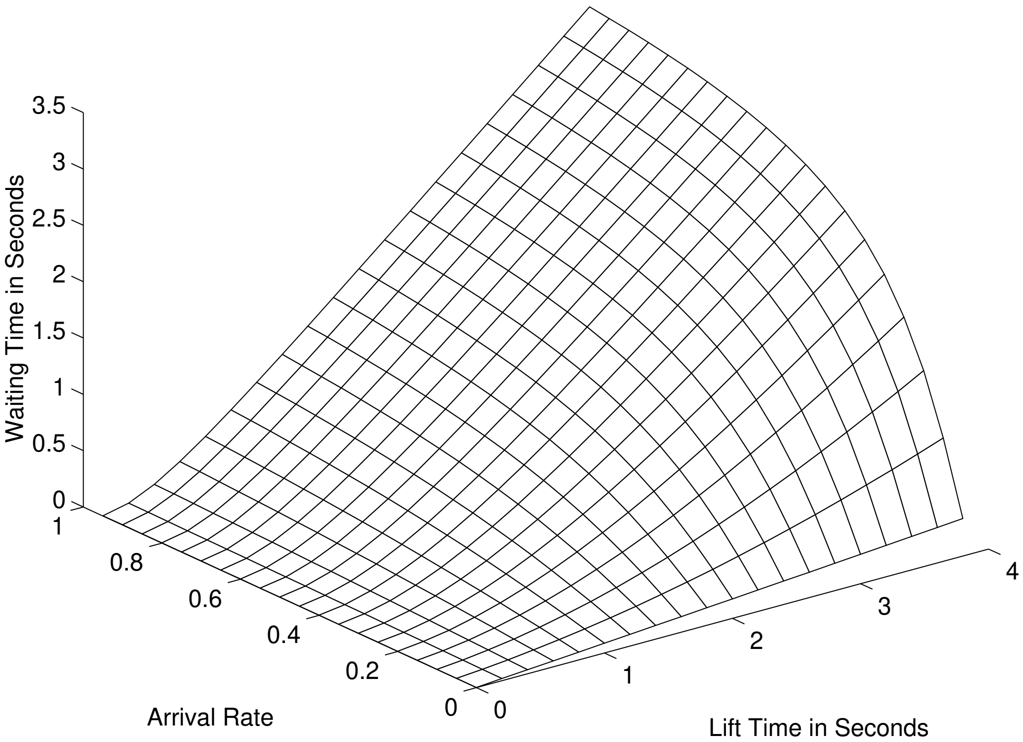

Figure 2: Performance of a Simple Two-Lift System

4 Analysing more complex systems

Our next task is to add more features of a realistic lift system. To achieve this goal we have considered two directions: increasing the number of floors and multi-person queues.

The main problem which we will encounter is an *exponential state explosion*, which is already known from model checking [11][1]. When components with a large number of states, acting fairly independently of the other components, are introduced, the number of states of the underlying process is increased dramatically. For example, taking two entirely independent lift systems, each identical to the two-lift system from the previous section (which has 144 states), yields $144 * 144 = 20736$ states. Analysis of such a model, by means of the PEPA workbench as it stands, would then be time-consuming.

Another issue that we have to deal with is the lack of parametrization and modularity in the workbench at present, as well as restrictions in the syntax.

Due to the large number of states we separately considered two orthogonal designs: a many-floor system with *up* and *down* call buttons and a realistic control element (we are in the process of analysing a two-lift system based on this), and a simplified lift with *up* buttons only, but many-person queues. In all cases, the trends observed were very similar to those described below.

4.1 Multi-floor lift

Initially, we aim to specify a three-floor system, which we can then *scale up* by increasing the number of middle floors. This means that a model where the interaction between the lift and the floors is made explicit is not acceptable. Thus, a naive extension of the simple two-lift system with an extra floor and lines like:

$$Lift_x \stackrel{def}{=} (UpOne, r_9).Lift_y + (UpTwo, r_{10}).Lift_z$$

will not be feasible.

What we require, instead, is some mechanism whereby the lift is passed up or down one floor at a time, but only stops at a floor when it needs to. We will need three elements: the control component (to determine where the lift should go from each stop), the lift (which we make oblivious to which

[1]The state explosion problem has been researched widely in the model checking community, leading to approaches that make the handling of 10^{20} binary states feasible. Tools for stochastic process algebras have not reached comparable maturity, but one would expect that some of the methods can be transferred from model checking to the SAPs.

floor it is on) and the floors. In addition, we shall need a 'translator' is needed to overcome the restriction of the workbench which limits the number of cooperating processes to two. It acts as a go-between passing messages between components.

These restrictions make the system unnecessarily complex, as the majority of the detail in each component has to do with the communications with the other elements, and it is not always possible to have these meet with intuition, due to the limitations of the language.

The lift itself is as follows.

$$
\begin{aligned}
Lift_0 &\stackrel{def}{=} (OneIn, \top).Lift_4 + (Go, l_0).Lift_0 \\
Lift_1 &\stackrel{def}{=} (LiftUp, l_1).Lift_2 + (LiftDown, l_1).Lift_3 \\
Lift_2 &\stackrel{def}{=} (Stop, l_2).Lift_0 + (LiftHere, l_3).Lift_5 \\
Lift_3 &\stackrel{def}{=} (Stop, l_2).Lift_0 + (LiftHere, l_3).Lift_6 \\
Lift_4 &\stackrel{def}{=} (InOK, l6).(Go, l0).Lift_1 \\
&\quad +(Call1, l_4).(Go, l_0).Lift_1 \\
&\quad +(Call2, l_4).(Go, l_0).Lift_1 \\
Lift_5 &\stackrel{def}{=} (Up, l_5).Lift_7 \\
Lift_6 &\stackrel{def}{=} (Down, l_5).Lift_7 \\
Lift_7 &\stackrel{def}{=} (Fail, l_2).Lift_0 + (LiftUp, l_1).Lift_2 + (LiftDown, l_1).Lift_3
\end{aligned}
$$

The lift is in state 0 when it is sitting on a floor. It may either take on passengers ($OneIn$), or it may be told to Go. If it takes on passengers (state 4), they will place a call, and then the lift will be told to Go as before ($InOK$ is needed to communicate with the floors). Once it has received this message, it is told which direction to head in via $LiftUp$ and $LiftDown$. At the next floor (state 2 or 3), it will either be told to $Stop$, in which case it returns to state 0, or it will be acknowledged with a $LiftHere$ communication. This latter possibility is for when the lift passes through a floor without stopping. In this case, it will tell the floor which direction it is heading in via Up and $Down$, and go to state 7. From here, the lift will be sent on its way via $LiftUp$ and $LiftDown$. It may also $Fail$ if it is attempting to go up from the top floor, or down from the bottom floor, and will stop (this is included as a safety catch).

The algorithm for the control element is as follows. We require a lift which will always go upwards, picking people up and dropping them off on the way, until there is no-one waiting on a floor above the lift, and no-one inside the lift wishes to get off on a higher floor. Then it will change direction and always go downwards, picking people up and dropping them off as before, until there are no calls or passengers waiting below, and so on repeatedly.

$$
\begin{aligned}
ControlUp &\stackrel{def}{=} (Ready0, \top).ControlU1 \\
&+(Ready1, \top).ControlU2 \\
&+(Ready2, \top).(FloorUp, c_0).ControlDown \\
ControlDown &\stackrel{def}{=} (Ready0, \top).(FloorDown, c_0).ControlUp \\
&+(Ready1, \top).ControlD0 \\
&+(Ready2, \top).ControlD1 \\
ControlU1 &\stackrel{def}{=} (CallOn1, \top).(FloorUp, c_0).ControlUp \\
&+(NoCallOn1, \top).ControlU2 \\
ControlU_2 &\stackrel{def}{=} (CallOn2, \top).(FloorUp, c_0).ControlUp \\
&+(NoCallOn2, \top).(Failure, c_2).ControlDown \\
ControlD_1 &\stackrel{def}{=} (CallOn1, \top).(FloorDown, c_0).ControlDown \\
&+(NoCallOn1, \top).ControlD_0 \\
ControlD_0 &\stackrel{def}{=} (CallOn0, \top).(FloorDown, c_0).ControlDown \\
&+(NoCallOn0, \top).(Failure, c_2).ControlUp
\end{aligned}
$$

The states $ControlUp$ and $ControlDown$ correspond to the overall general heading of the lift. In other words, when the lift is 'always going upwards', the control element will be in state $ControlUp$. When the lift stops at a floor, the control element receives a $Ready$ communication from that floor. It then looks through the states above or below the lift, depending on the direction the lift is heading in, for a call. If it finds one, it tells the floor to send the lift on. If it doesn't, it changes the overall lift direction.

This is best demonstrated with an example. Suppose the lift is heading downwards and has arrived at the ground floor. Then the control is in state $ControlDown$ and receives a $Ready0$ communication, which puts it in state $ControlDown2$. From this state, it tells the floor to send the lift down ($FloorDown$), and changes direction by going to state $ControlUp$. It is obviously not possible for the lift to go down, and when we come to the floor component we will see that it treats this message as a failure, and sends the $Ready0$ message again. This time the control is in the $ControlUp$ state. It receives the message, and goes into state $ControlU1$.

Now suppose there is a call on floor 1. Then the control recieves the $CallOn1$ communication and goes into state $ControlU2b$, from where it tells the floor to send the lift up, and goes back to state $ControlUp$. Conversely, suppose there is no call on either floor 1 or floor 2. Then the control component goes via $NoCallOn1$ and $NoCallOn2$ to state $ControlU2c$. From here it reports a $Failure$ to the floor, and changes direction again, to $ControlDown$. Note that there is some redundancy here, since the $Failure$ communication achieves the same effect as the $FloorDown$ message when it is sent to the ground floor. This is a side-effect of the development process, and is needed to ensure correctness.

Each floor can be loosely divided into two sections. The first section, given below (for the ground floor), models the behaviour of the floor when the lift is not there.

$$\begin{aligned}
Floor0_0 &\stackrel{def}{=} (Arrival, f_0).Floor0_1 + (Call0, \top).Floor0_2 \\
&\quad +(Ex0, \top).Floor0_3 + (NoCallOn0, f_1).Floor0_0 \\
Floor0_1 &\stackrel{def}{=} (CallOn0, f_1).Floor0_1 + (Call0, \top).Floor0_1 \\
&\quad +(Ex0, \top).Floor0_4 \\
Floor0_2 &\stackrel{def}{=} (CallOn0, f_1).Floor0_1 + (Call0, \top).Floor0_1 \\
&\quad +(Arrival, f_0).Floor0_1 + (Ex0, \top).Floor0_5 \\
Floor0_3 &\stackrel{def}{=} (Arrival, f_0).Floor0_4 + (Call0, \top).Floor0_5 \\
&\quad +(LiftHere, \top).Floor0_8 + (NoCallOn0, f_1).Floor0_3 \\
Floor0_4 &\stackrel{def}{=} (CallOn0, f_1).Floor0_4 + (Call0, \top).Floor0_4 \\
&\quad +(Stop, \top).Floor0_7 \\
Floor0_5 &\stackrel{def}{=} (CallOn0, f_1).Floor0_5 + (Call0, \top).Floor0_5 \\
&\quad +(Arrival, f_0).Floor0_4 + (Stop, \top).Floor0_6
\end{aligned}$$

This part of the floor component was designed after the following paradigm. The floor can have three things: it can have an *arrival*, it can receive a *call* (from someone inside a lift, to get out on that floor), and it can have an *expect lift* token (indicating that the lift is about to arrive at the floor). It must also respond to queries from the control about whether or not it has a call (in this case, if it has an arrival it will respond affirmatively).

The rest of the floor description is concerned with the control of the lift as it passes through the floor, or stops at the floor and moves off again.

$$\begin{aligned}
Floor0_6 &\stackrel{def}{=} (Ready0, f_2).Floor0_9 \\
Floor0_7 &\stackrel{def}{=} (OneIn, f_4).(InOK, \top).Floor0_6
\end{aligned}$$

$$\begin{aligned}
Floor0_8 &\stackrel{def}{=} (Up, \top).(ExOn1, f_3).(Sent, \top).(LiftUp, \top).Floor0_0 \\
&\quad +(Down, \top).(Fail, \top).Floor0_6 \\
Floor0_9 &\stackrel{def}{=} (FloorUp, \top).(ExOn1, f_3).(Sent, \top) \\
&\quad .(Go, \top).(LiftUp, \top).Floor0_0 \\
&\quad +(FloorDown, \top).(Ready0, f_2).Floor0_9 \\
&\quad +(Failure, \top).Floor0_6
\end{aligned}$$

State 6 is for when the lift is stopping on the floor. It sends the *Ready* communication to the control which we discussed earlier. State 7 lets arrivals into the lift and then behaves as state 6. State 8 is for a lift passing through. If it receives an *Up* message from the lift, it will pass the lift on via the $ExOn1$, *Sent* and *LiftUp* messages. These respectively tell the next floor to expect the lift, wait for acknowledgment of this message (lest the lift be sent on before the next floor is expecting it, and trace of the lift is lost), and to actually send the lift on. If it receives a *Down* message, it fails and stops the lift (again, this should not be possible in practice). State 9, for the stopped lift, receives instruction from the control element, and subsequently behaves in much the same way as state 8.

For completeness, we include now the translation component, although its operation should be self-explanatory. We also give the cooperation set underneath.

$$\begin{aligned} T = {} & (ExOn0, \top).(Ex0, t_0).(Sent, t_1).T \\ & +(ExOn1, \top).(Ex1, t_0).(Sent, t_1).T \\ & +(ExOn2, \top).(Ex2, t_0).(Sent, t_1).T \end{aligned}$$

$$((((Floor0_6 \underset{\emptyset}{\bowtie} Floor1_0) \underset{\emptyset}{\bowtie} Floor2_0) \underset{\mathcal{P}}{\bowtie} T) \underset{\mathcal{Q}}{\bowtie} Lift_0) \underset{\mathcal{R}}{\bowtie} ControlUp$$

where $\mathcal{P} = (ExOn0, ExOn1, ExOn2, Ex0, Ex1, Ex2, Sent)$,
$\mathcal{Q} = (OneIn, Go, LiftUp, LiftDown, Stop, LiftHere,$
$Call0, Cal1, Call2, Up, Down, Fail)$,
and $\mathcal{R} = (CallOn0, CallOn1, CallOn2, Ready0, Ready1, Ready2, Failure,$
$NoCallOn0, NoCallOn1, NoCallOn2, FloorUp, FloorDown)$

This model, and the thought processes behind its design, has been described in detail, to try and demonstrate the inherent difficulty in describing such a lift in PEPA while maintaining correctness assurance.

4.2 Analysing the multi-floor lift

The analysis of mean waiting time is performed similarly to the previous model, except that now the model has 658 states in the case of three floors. The results are given in Figure 3.

We note first that this once more exhibits the trends we have seen throughout. We also note that this is somewhat *faster* than our previous attempt, which had 860 states and did not include the *InOK* communication. This was quite unexpected, but can be explained as follows. Suppose, for exam-

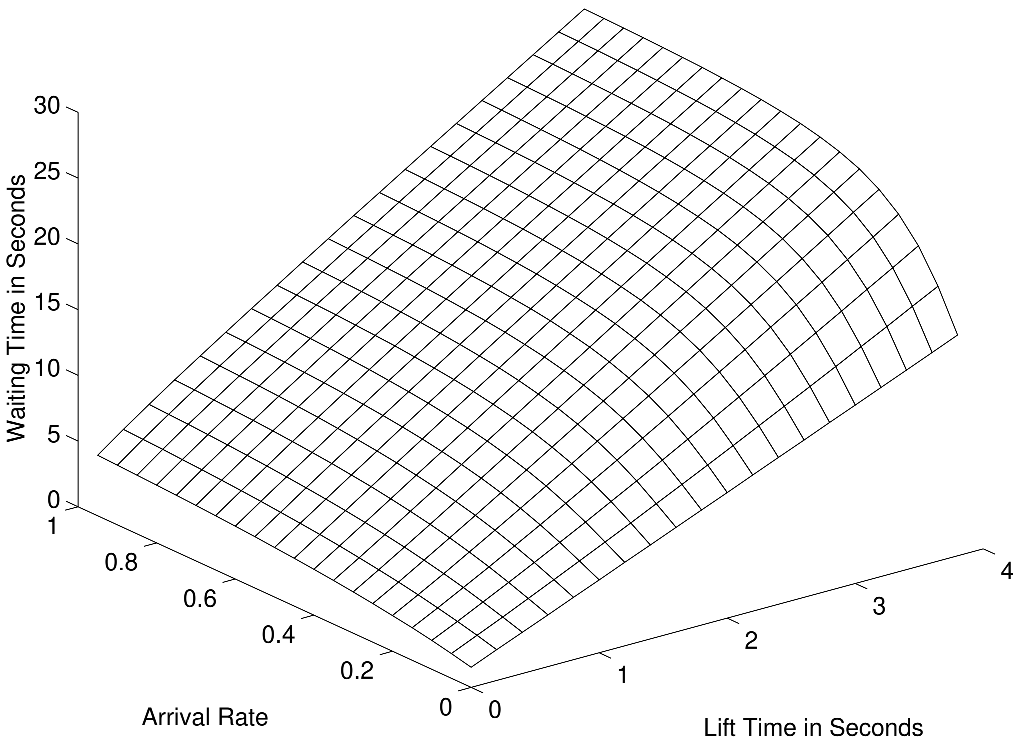

Figure 3: Performance of the Multi-Floor Lift

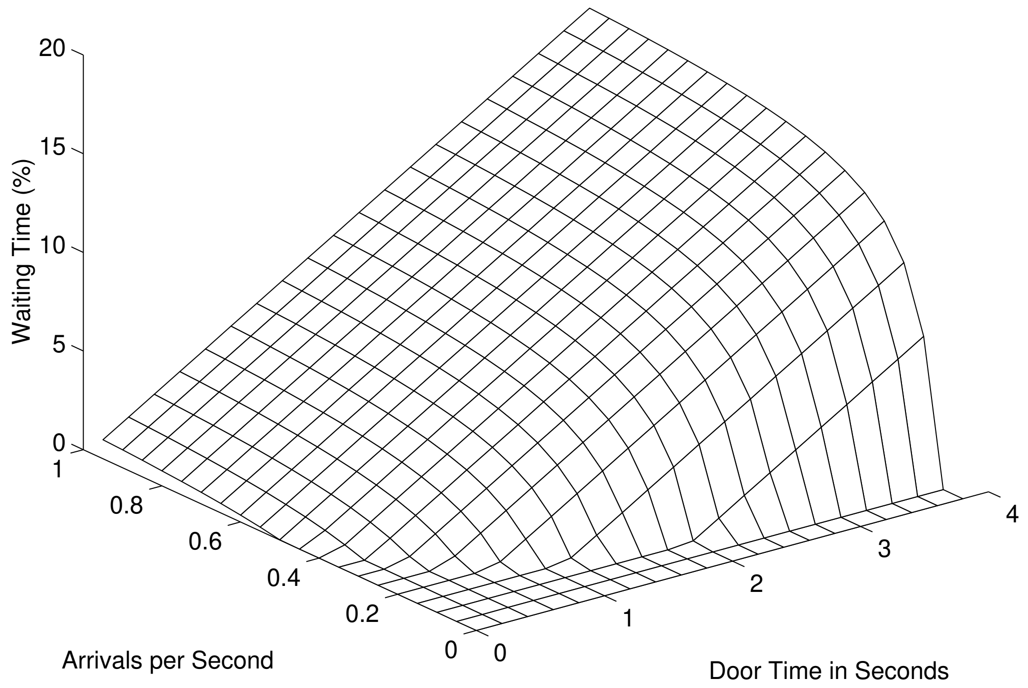

Figure 4: Graph of Engineer's Lift Analysis

ple, that the lift is on the first floor, heading up, and someone has just got into it. If the system is as lift 4, then the lift could be sent down, even if the person who just got in wanted to go up. This is because the lift can finish communicating with the control before the call communication from the person inside is placed. On the other hand, if the system is as lift 5, then the lift will always go up in these circumstances, as the call from the passenger will be placed before the control is consulted.

4.3 Comparing with traditional engineering techniques

We are now in a position to compare our results with those we would have obtained using tradition engineering methods. In Figure 4, we have applied an equation based method detailed in Barney [2] to a similar system to the lift above.

We notice, firstly, that the two methods produce models exhibiting remarkably similar trends (they are not likely to be numerically comparable as they model slightly different types of lift). However, the engineering approach is certainly not as accurate for low arrival times and lift speeds. It suggests that the average waiting time can be zero in such cases, which is clearly not reasonable. It is likely that the PEPA approach is more numerically accurate over all values of arival rate and lift speed, but this is hard to establish in general in this case.

5 Conclusions

Our experience with the PEPA workbench was largely positive, in the sense that we did not encounter any flaws and the results obtained were consistent, while at the same time more accurate, than those obtained by the traffic analysis methods [2]. While the derivation of the model is a non-trivial task, we found that once this has been obtained, introducing desired cost or performance measures is straightforward, and so a variety of analyses can be done on the model.

We did, however, find a number of features difficult in the tools as it stands. We list them for the benefit of future developers of the tool.

First, we found that high level of confidence in the correctness of the model was required. This is because there are no means for *tracing execution* (whether 'running' the model or unfolding the multi-graph), *debugging*, emphdeadlock detection, or (automatic or semi-automatic) *verification* of properties. Some work on *code generation* [6] may offer partial help in this case, but also the development of the necessary methodologies and tools, based e.g. on the recent work of [1], seems a worthwhile long-term goal.

Two other issues that we would like to raise were the lack of *parametricity* and *modularity*. The former would have allowed us to specify a multi-floor lift by describing the ground and top floors, and then the middle floor could be indexed by the parameter i, where i could be instantiated to some value, say 5, before analysis. The latter could be used e.g. to replace a lift control algorithm with another one. Clearly defined interfaces are needed for this purpose, and also equivalence / simulation testing to ensure that substitution can be made.

The handling of models with a large number of states was also uncessarily tedious because defining performance measures *directly* at the level of PEPA, where attaching rewards to symbolic process names is possible, is not allowed at present. More work in this direction, as well as towards full exploitation of compositionality, is needed, although it is difficult to imagine that one can dispose of Matlab completely.

References

[1] A. Aziz and K. Sanwal and V. Singhal and R. Brayton. Verifying Continuous Time Markov Chains. In R. Alur and T. Henzinger, editors, *Proceedings of Computer Aided Verification (CAV)*, volume 1102 of *Lecture Notes in Computer Science*. Springer Verlag, 1996.

[2] G.C.Barney. Theoretical design aspects of lift traffic, In G.C.Barney, editor, *Elevator Technology*, Ellis Horwood, 1986.

[3] M.Bernardo, L.Donatiello and R.Gorrieri. Integrating performance and functional analysis of concurrent systems with EMPA, *Technical Report, Department of Computer Science, University of Bologna, 1995.*

[4] 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, volume 794 of *LNCS*, Springer-Verlag, 1994. Pages 353-368.

[5] S.Gilmore, J.Hillston, D.R.W.Holton, and M.Rettelbach. Specifications in a Stochastic Process Algebra for a Robot Control Problem. *International Journal of Production Research*, 1995.

[6] S.Gilmore, J.Hillston, and D.R.W.Holton. From SPA models to programs. In *Proceedings of PAPM96*, to appear.

[7] N. Gotz, U. Herzog and M. Rettelbach. TIPP - a language for timed processes and performance evaluation. In U.Herzog and M.Rettelbach (editors). Proceedings of the 2nd workshop on performance modelling, *Erlangen, Germany, 1994.*

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

[9] J.Hillston and U.Mertsiotakis. A simple time scale decomposition technique for stochastic process algebras, *The Computer Journal*, vol.38, no.3, 1995.

[10] D.R.W.Holton and J.P.N.Glover. An SPA Performance Model of a Production Cell. Preprint, 1996.

[11] K. L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, 1993.

[12] R.Milner. Communication and concurrency, *Prentice Hall, 1989.*