Towards reliable software performance modelling using stochastic process algebra

Nigel Thomas and Jeremy Bradley
Research Institute in Software Evolution,
University of Durham, UK

Abstract

The aim of any modelling exercise is to develop a better understanding of the system that is being studied, however it is unfortunately not always easy to understand abstract performance models or the metrics that are derived from them. In this tutorial paper we focus on the facilitation of more reliable performance prediction through better understanding of models and solutions in the context of a stochastic process algebra. As well as being confident that the model is accurate, the designer must also have confidence in the measures derived. Partly this is a matter of ensuring that the correct measures are derived, but also that those measures are sufficiently bounded to be applicable.

1. Introduction

Increasingly, practitioners in mainstream software engineering are recognising the need for qualitative predictions of the services that their software systems provide. This is especially evident in the growing fields of WWW based applications and component based systems where there is a large amount of interaction between system elements and non-functional information, such as timing, which is crucial to usability. Traditional, post-development, code-based metrics are no longer adequate indicators of a products ability to maintain a market position and a wide range of analysis techniques typically need to be applied. Performance modelling provides one such class of evaluation techniques. At the same time it is recognised that future software development must not lead to large, inflexible, legacy type systems. A far greater emphasis has therefore been put on understanding the structure of existing complex software systems, applications in development and the impact of changes. A valuable set of visualisation techniques has been developed over recent years to support comprehension of such systems [31,32,45,46].

Because of the properties of compositionality, parsimony and expressiveness, the possibility of including domain information and the ability to automatically derive performance measures, stochastic process algebra (SPA), such as PEPA [23], are an extremely good modelling paradigm for a wide variety of different domains. In addition, it has been successfully demonstrated that performance models can be automatically derived from design specifications in the Unified Modelling Language (UML) [28,29,30,33,35,37,38] and that SPA models can be related directly to code [18,39]. These developments have enabled performance models, and in particular SPA models, to be studied by software engineers with little or no experience of stochastic modelling.

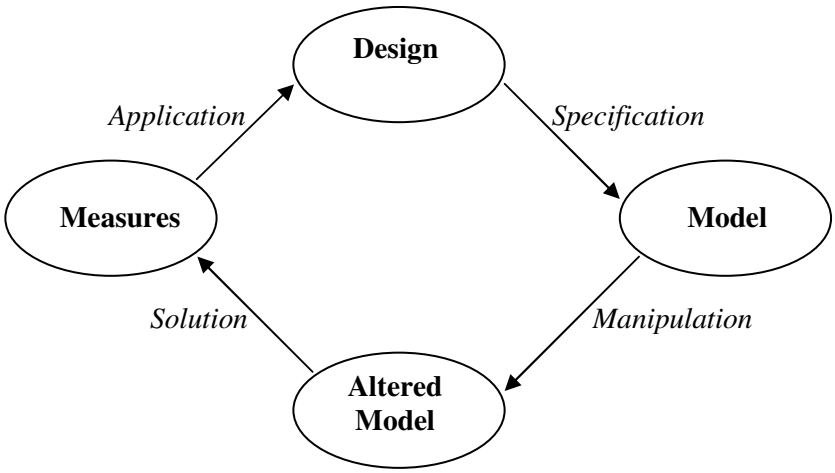

Figure 1.1: A typical performance modelling cycle

Figure 1.1 shows a typical modelling cycle. The process starts with an elementary design, or system specification, which is then used to specify a performance model of the system. To facilitate easier solution, or in order to include additional features such as reward structures the model may be manipulated into an alternative form, which is then solved to derive performance measures. Since measures are insufficient in themselves to fully analyse the system, they must be applied, or interpreted, to facilitate the evaluation of the design and development of improvements. Clearly, it is vital that the designer has confidence in the measures that are derived in order that any changes made to the design are valid. Measures taken from the model must therefore be related to properties of the system rather than the model and must be accurate and predictable. In addition the designer must have confidence that the model which is solved accurately depicts the behaviour of the system which is being designed.

By using an established design notation, such as UML, and automatically deriving a performance model, in SPA for example, the designer can have some confidence that there is a strong relationship between the model and the design. The model specification process, as presented by Pooley and King [37], relies on identifying the observable behaviour and interaction between objects in the design specification in order to identify the action sequences and synchronisation that need to be included in the model. In addition, because the objects in the design and the model are the same, the design notation may be used to explicitly define performance measures in terms of these objects. The quality of the measures derived is determined by the solution techniques employed, variance analysis allows the degree of reliability of measures to be predicted and variance reduction improves confidence of those measures [8].

In the following section we cover a basic introduction to performance modelling with stochastic process algebra, aimed at experienced performance engineers with little or no previous knowledge of stochastic process algebra. We present several examples covering the informal translation of models from stochastic Petri nets and queueing networks to stochastic process algebra. In Section 3 visual techniques from program comprehension are applied to performance models to aid understanding and improve confidence that the model accurately reflects design. Some visual representations are also used to derive design views in UML to provide meaningful feedback in a modelling cycle.

2. Stochastic Process Algebra

Stochastic process algebra, extensions to classical process algebra such as CCS [34] and CSP [26], add several attractive features to performance modelling:

- Compositionality
- Formality
- Abstraction

From a specification point of view these features allow models to be constructed.

- Concisely
- With their meaning incorporated explicitly
- In a way most natural to the modeller

They also allow complex models to be analysed and solved efficiently and automatically.

A number of stochastic process algebra have been defined, among them

- PEPA [23](University of Edinburgh, UK)
- TIPP [19](University of Erlangen, Germany)
- EMPA [4](University of Bologna, Italy)
- SPADE (Imperial College, London, UK)

Although there are semantic differences between these process algebra the modelling differences are not great, the exception being SPADE which allows generally distributed actions and model solution by simulation. In this paper we concentrate on PEPA, although most of our argument is clearly applicable the other algebra with some small modification.

2.1. PEPA

PEPA is a Markovian process algebra, meaning all events occur in response to exponentially distributed delays. Models are defined as the interaction of components which engage, singly or multiply in activities. Each component may be atomic or composed of other components. Each activity $a \stackrel{\text{def}}{=} (\alpha, r)$ has an action type α and is exponentially distributed with rate r or passive with distinguished rate $\top$. There are a small number of combinators which are used to define models.

Constant: $A \stackrel{\text{def}}{=} B$ used to assign names to components.
Prefix: $(\alpha, r).P$ the component engages in action α at rate r and subsequently behaves as P.
Choice: $P+Q$ the component behaves as either P or Q, the choice is determined by a race condition on the first action to complete.
Hiding: P/L the actions in the set L are not visable outside P and cannot be shared.
Co-operation: $P \underset{L}{\bowtie} Q$ the components proceed independently with any activities whose types do not occur in the cooperation set L. Activities with action types in the set L are only enabled in $P \underset{L}{\bowtie} Q$ when they are enabled in both P and Q. In PEPA the shared activity occurs at the rate of the slowest participant. If an activity has an unspecified rate in a component, the component is *passive* with respect to that action type.

2.2. Queueing Network Models

PEPA can be used to specify many models that can also be specified using other modelling paradigms. It is a simple matter to describe queues in PEPA using a state-based approach, it is also possible to use a job based approach. By constructing PEPA components of queues, large queueing network models can be formed relatively easily in PEPA, although some care is needed to ensure correct naming of shared actions.

Figure 2.1 shows a state-based model of an M/M/1/N queue. The model is defined as the interaction of two components: the queue component and an arrival/service component. The queue component represents the number of jobs present in the queue and determines what *arrival* and *service* actions are possible. We define the queue to be passive and sharing actions *arrival* and *service* with the other component, S. When behaving as $Queue_0$ no *service* is possible (obviously there are no jobs to serve) as the *service* is not present here and it is a shared action, in this instance the component *S* cannot fire the service action. Similarly the arrival action is blocked when the queue behaves as $Queue_N$ (the queue is full).

$$
\begin{aligned}
Queue_0 &\stackrel{\text{def}}{=} (arrival, \top).Queue_1 \\
Queue_j &\stackrel{\text{def}}{=} (arrival, \top).Queue_{j+1} + (service, \top).Queue_{j-1} \quad , \quad 1 \leq j \leq N-1 \\
Queue_N &\stackrel{\text{def}}{=} (service, \top).Queue_{N-1}
\end{aligned}
$$

$$
\begin{aligned}
S &\stackrel{\text{def}}{=} (arrival, \lambda).S + (service, \mu).S \\
S &\underset{\{arrival,\ service\}}{\bowtie} Queue_0
\end{aligned}
$$

Figure 2.1: An M/M/1/N queue in PEPA

Clearly it would be simple to model this system as a single component or split the arrival and service elements of *S* into separate components. Within this simple set up it is easy define interrupted services or Markov-modulated arrival processes. In Figure 2.2 the component *S* has been replaced by a two phase process. When behaving as S_1 arrivals and services occur as previously, however in S_0 the service process is blocked and the arrivals occur at a different rate. The behaviour is switched between the two by the actions *sleep* and *wakeup*, which are not shared with the queue.

$$
\begin{aligned}
S_1 &\stackrel{\text{def}}{=} (arrival, \lambda_1).S_1 + (service, \mu).S_1 + (sleep, \xi).S_0 \\
S_0 &\stackrel{\text{def}}{=} (arrival, \lambda_0).S_0 + (wakeup, \eta).S_1 \\
S_1 &\underset{\{arrival,\ service\}}{\bowtie} Queue_0
\end{aligned}
$$

Figure 2.2: Service interruptions and variable arrival rate

When considering models of multiple queues, where jobs may progress from one queue to the next, the *service* action in one component must act as an *arrival* action in another. Such a situation is illustrated in Figure 2.3. In PEPA this kind of consequence must be defined using the same named action in both components, which can be confusing (a renaming operation would be useful). For example, in Figure 2.3 the action *service1* is used to mean both a departure from the first queue and an arrival at the second, they are clearly different ways of viewing the same event. In the second queue we have used two kinds of service event, *leave* causes the job to leave the system and *feedback* which triggers an arrival into the first queue (the counterpart to *service1*).

A great many queueing behaviours can be modelled using PEPA (see [5,41,42]), although the use of a textual interface in the PEPA Workbench does not always make such models easy to define or interpret. To this ends a queueing interface for PEPA, called Dr PEPA, has been built by Jim Newton at the University of Durham. Dr PEPA accepts PEPA scripts and parses them to see if they conform to a known way of specifying a queue (only a state-based representation is handled). If the model parses successfully the tool draws a graphical representation of the model, using colour to depict the different components in both the script and the drawing. The tool also supports simple navigation mechanisms to locate components

in the script or drawing and an edit an update function from the script to the drawing. As yet Dr PEPA does not support graphical editing of PEPA scripts. A sample screen shot from Dr PEPA is shown in Figure 2.4.

$$
\begin{aligned}
Queue1_0 &\stackrel{\text{def}}{=} (arrival, \lambda).Queue1_1 + (feedback, \top).Queue1_1 \\
Queue1_j &\stackrel{\text{def}}{=} (arrival, \lambda).Queue1_{j+1} + (service1, \mu_1).Queue1_{j-1} \\
&\quad + (feedback, \top).Queue1_{j+1} \;\;,\;\; 1 \leq j \leq N-1 \\
Queue1_N &\stackrel{\text{def}}{=} (service1, \mu_1).Queue1_{N-1} + (feedback, \top).Queue1_N \\
\\
Queue2_0 &\stackrel{\text{def}}{=} (service1, \top).Queue2_1 \\
Queue2_j &\stackrel{\text{def}}{=} (service1, \top).Queue2_{j+1} + (feedback, q\mu_1).Queue2_{j-1} \\
&\quad + (leave, (1-q)\mu_1).Queue2_{j-1} \;\;,\;\; 1 \leq j \leq N-1 \\
Queue2_N &\stackrel{\text{def}}{=} (service1, \top).Queue2_N + (feedback, q\mu_1).Queue2_{N-1} \\
&\quad + (leave, (1-q)\mu_1).Queue2_{N-1} \\
\\
Queue1_0 &\underset{\{service1,\, feedback\}}{\bowtie} Queue2_0
\end{aligned}
$$

Figure 2.3: Two M/M/1 queues in tandem

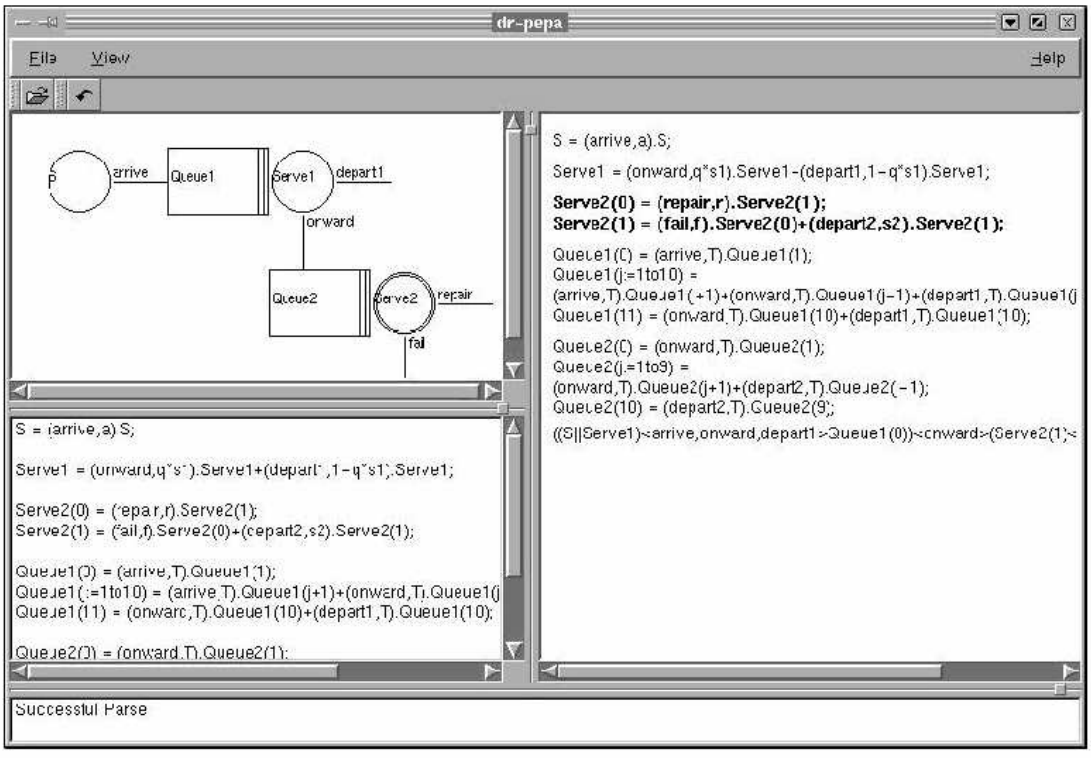

Figure 2.4: "Dr PEPA" queueing model interface for PEPA

2.3. Stochastic Petri Nets

Of course, a graphical based formalism for performance modelling has been available for sometime in the form of stochastic Petri nets [1]. In general it is easy to informally translate models between stochastic process algebra and stochastic Petri nets, although there are no immediate actions in PEPA. A comparison between PEPA and GSPN has been made by

Donatelli et al [14,15], who defined a set of Petri net equivalents to the basic PEPA constructs, shown in Figure 2.5.

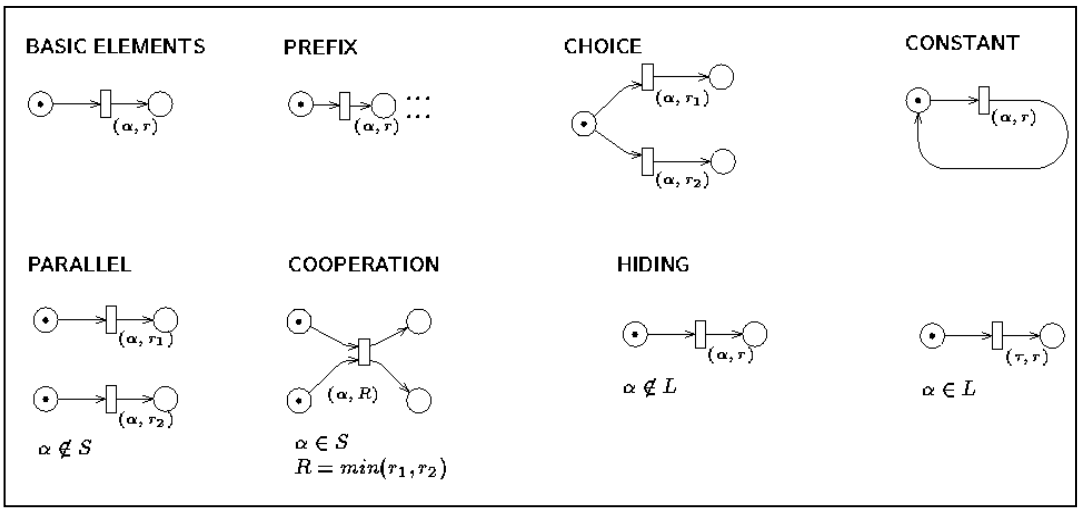

Figure 2.5: Basic constructs in PEPA and GSPN (From Donatelli et al [15])

This equivalence equates GSPN, place and PEPA constants and GSPN transitions are PEPA actions. A more formal approach has been taken by Ribaudo [40], and Bernado et al in integrating tool support for EMPA and GSPN by defining EMPA in terms of GSPN semantics [2,3].

Figure 2.6 shows a simple model of a processor using a resource. The processor does an initial *think* action before requiring to *use* the resource. Following completion of the *use* action the processor returns to thinking and the resource must go through an *update* phase before being available for use once more. If the *think* action completes before the corresponding *update* then the processor must wait. The PEPA and GSPN specifications shown give rise to models with identical Markov chains.

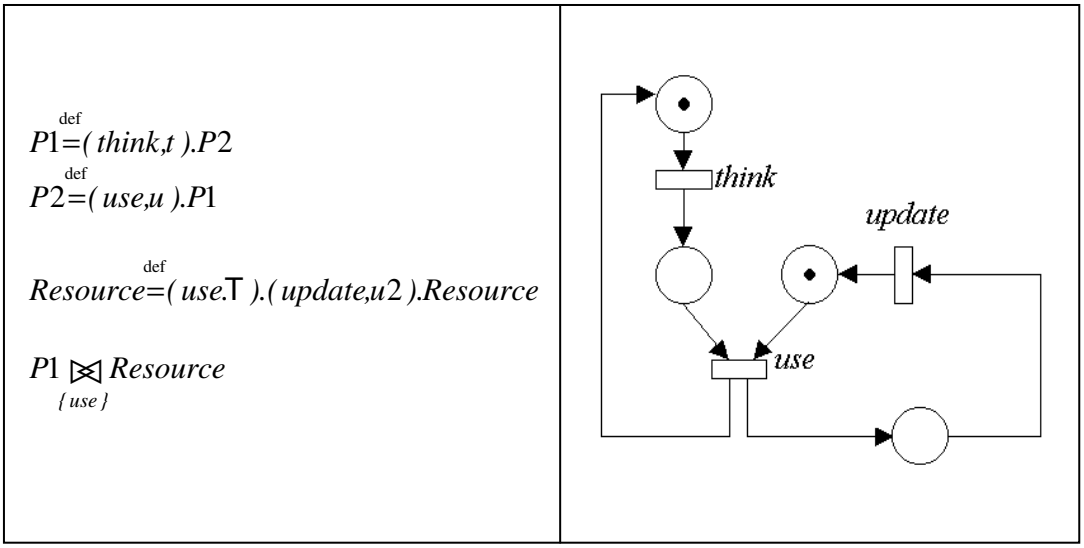

Figure 2.6: A simple PEPA model of resource usage with an equivalent GSPN representation

A slightly more complex example is given in Figure 2.7. Here the processor must *get* and subsequently *release* a lock on two resources, a *Bus* and *Memory*, following *release* the two resources are immediately available once again. Of course, there is little point in locking a

resource if only one process has access to it. Figure 2.8 shows an extension to the previous model with two competing processes.

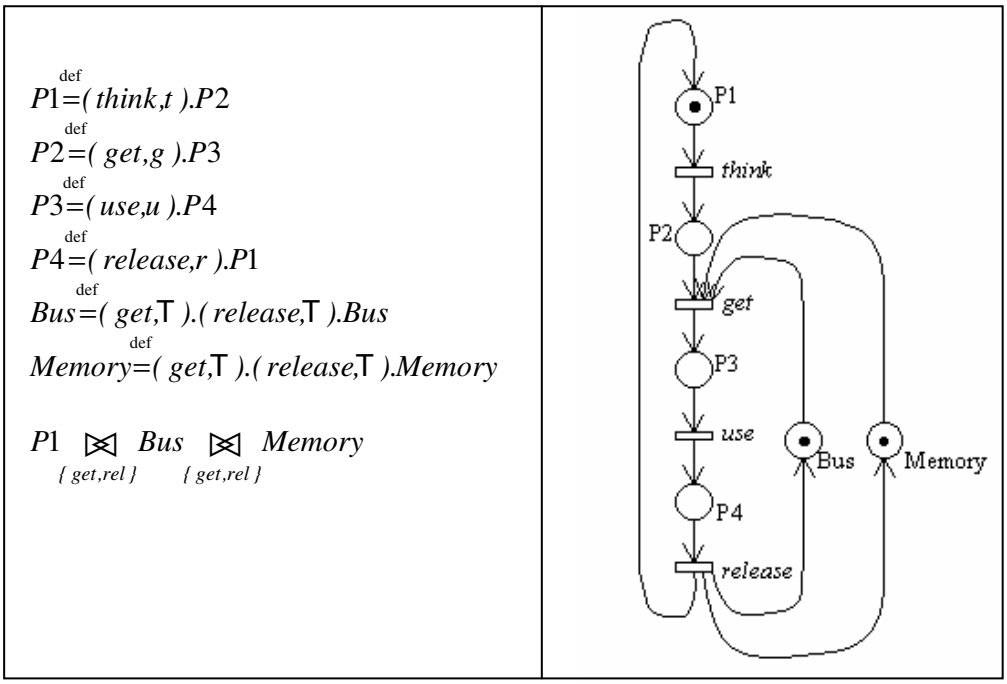

Figure 2.7: PEPA model of resource locking with an equivalent GSPN representation

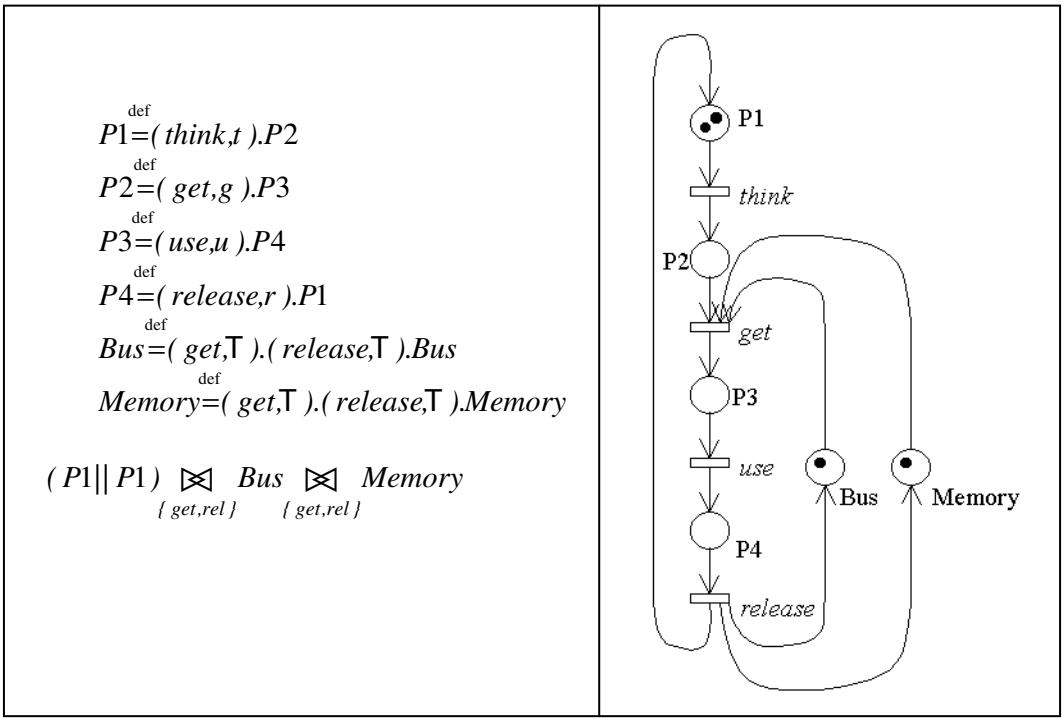

Figure 2.8: Multiple instances vs. multiple tokens

On first inspection the two methods for extending the model, adding an extra token in GSPN and adding a second instance of the processor component, appear to be equivalent. However, the two methods do not give rise to identical models. The distinction can be seen if we consider the firing of a *think* action/transition. In the GSPN model a token moves from *P1* to *P2*, whichever token moves the subsequent markings are identical. In the PEPA model

however, either the first instance processor performs the action or the second instance does, leading to two distinct successors:

$$(P2 \| P1) \underset{\{\text{get,rel}\}}{\bowtie} Bus \underset{\{\text{get,rel}\}}{\bowtie} Memory$$

and,

$$(P1 \| P2) \underset{\{\text{get,rel}\}}{\bowtie} Bus \underset{\{\text{get,rel}\}}{\bowtie} Memory$$

In practice most practical performance measures will not distinguish between these two cases, however, it should be noted that they are not identical.

3. Understanding performance models

There are several reasons for a system designer not to trust the results from a performance model:

- Don't understand the model behaviour.
- Measures don't relate to the design.
- Wrong metrics.
- Wrong method of solution (not always obvious!).
- Incomplete information.

These problems generally arise because the modelling and design tasks are divorced and the system designer only sees the results of the modelling study and does not have in depth knowledge of stochastic modelling. An over reliance on automated methods can mean that a non-expert modeller solves a model that has been over simplified or with inappropriate assumptions made.

In the analysis of a PEPA model a derivation graph is formed in order to compute the states of the underlying Markov chain. This graph is not presented explicitly to the modeller by the PEPA Workbench tools [11,17], rather it is an important mechanism used to study the PEPA model and derive a numerical solution. This type of graph structure is extremely similar to call graphs used in program comprehension and as such the same set of tools can be used to visualise them. Figure 3.1 shows the PEPA specification for a shared resource system, adapted from Hillston [23], and Figure 3.2 shows its derivation graph.

$$P \stackrel{\text{def}}{=} (use, r_1).(task, r_2).P$$
$$R \stackrel{\text{def}}{=} (use, \top).(update, r_3).R$$
$$(P \| P) \underset{\{use\}}{\bowtie} R$$

Figure 3.1: PEPA specification of a shared resource model.

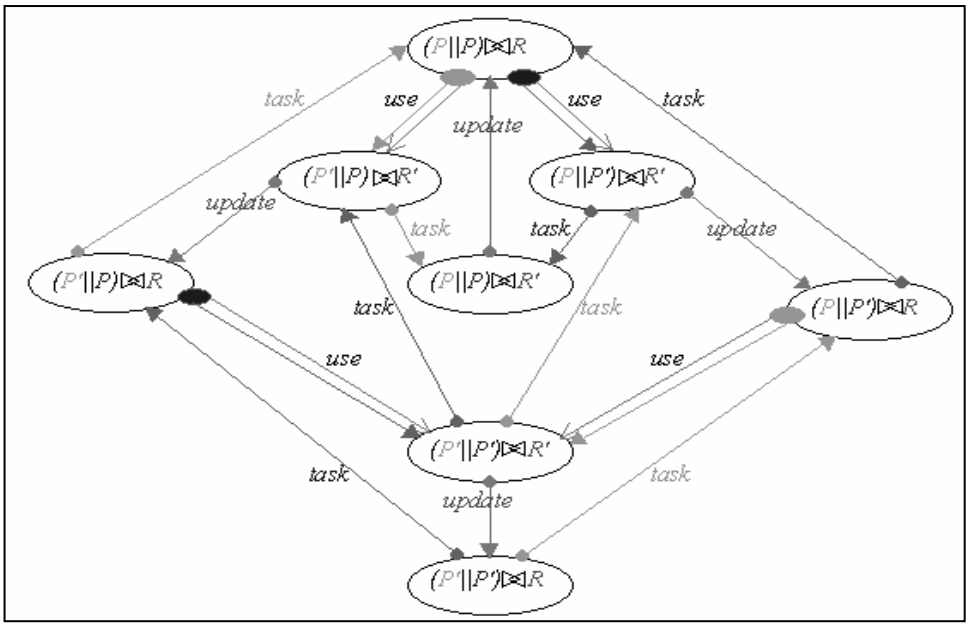

Figure 3.2: A coloured derivation graph

The graph is enhanced using colour to show what agents are participating in the actions and are subsequently evolved. In the case of shared activities multiple coloured arrows indicate the participating agents, filled head arrows indicate active actions and open headed arrows indicate passive actions. Clearly this is a small model and it is possible to view the entire model and derivation graph without difficulty, however, the inclusion of the graph and the addition of colour add to the ease by which non-experts may understand the evolution of this model. Given a larger state space the size of the full derivation graph in this form quickly becomes unmanageable. Several possibilities exist for handling problems of scale in static 2D representations, briefly these may be summarised as:

- **Elimination**. Nodes with only one subsequent action may be removed and actions combined.
- **Aggregation**. Similar nodes, or nodes relating only to internal activities of agents, are combined.
- **Decomposition**. Each individual component is viewed in isolation, although it behaves in the same way as in the full model.
- **Highlighting**. Although the entire graph is displayed, certain arcs and nodes are emphasised (such as those pertaining to the evolution of a particular component). Highlighting can be used to show sequences of independent behaviour.
- **Layering**. The entire graph is viewed in simple form with gross aggregation of nodes. Aggregated nodes may be selected for expansion within the full graph or as a separate graph.
- **Abstraction**. The entire graph is viewed in simple form with gross aggregation of nodes; the detailed behaviour within the aggregated nodes may be shown in miniature, as if from a distance.
- **Windowing**. The entire graph is displayed within a sliding window, a miniature representation of the graph may be used to show the position of the window to aid navigation.

Using the above example there is little scope for elimination, except for removing the node $P||P \bowtie R'$ and compounding the subsequent *update* preceding *task* actions to give two *task*+*update* arcs to $P||P \bowtie R$ from $P'||P \bowtie R'$ and $P||P' \bowtie R'$ respectively. There is scope for aggregation by exploiting the symmetry of the graph.

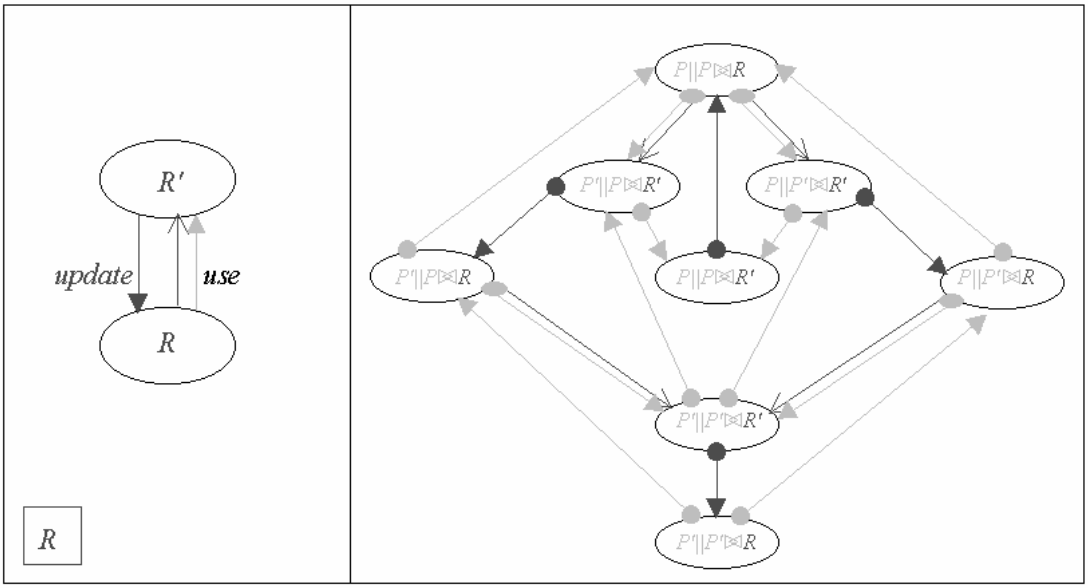

Figure 3.3: Decomposed view and highlighted derivation graph for resource component, *R*.

Figure 3.3 shows the resource component in isolation and highlighted within the derivation graph. Within this simple example there is only a limited benefit from such representations over the coloured derivation graph. The alternate actions, *use* followed by *update*, are clearly evident in the highlighted derivation graph, showing that the resource component in the model behaves as expected. In larger models, with more complex components, sequences of independent actions can be highlighted

Thus far we have presented information in a solution oriented manner, that is, with nodes being equivalent to states in the underlying Markov process. Nodes are the primary conceptual foci of graph representations and so in viewing a derivation graph, the user places state as the principal model object. However, designers generally have little or no concept of state, but rather tend to consider software systems as collections of functions or activities performed according to certain ordering constraints. The translation from the state based derivation graph to a functional, or activity oriented, view is a simple matter of exchanging nodes and arcs, so that the nodes represent actions and the arcs represent pre- and post-conditions. Alternatively we could present such a view using call stack representations which have been used successfully in program visualisation [46].

In the discussion above we have proposed representations from performance and behavioural viewpoints that aid understanding of the evolution of the entire model. We now concentrate on views of the high level objects and the interfaces that are formed between them by synchronised actions. Visualisations such as these allow the designer to observe that the objects in the model correspond to objects in the design and that the way in which those objects interact is similarly represented. At this level of abstraction it is not necessary for us to present the detailed sequences of behaviour that lead to these interactions, rather how the interaction is facilitated (the shared action) and possibly any pre-conditions. Much of the necessary information for this is contained in statements containing the cooperation combinator, for example the following statement taken from a model by Holton [2]:

$$Workcell \stackrel{\text{def}}{=} Robot \underset{S_1}{\bowtie} ((Belt \underset{S_2}{\bowtie} Table) \| Press \| (DBelt \underset{S_2}{\bowtie} Crane))$$

$$S_1 = \{ ready_to_pick , unload_blank , DBelt_ready , load_blank \}$$

$$S_2 = \{ ready_to_put \}$$

$$S_3 = \{ blank_ready \}$$

Figure 3.4: PEPA model element of a workcell specification

Robot, *Belt*, *Table*, *Dbelt*, *Crane* and *Press* are the initial agents of each of the five components representing a robot arm, a feed belt, an elevating table, a deposit belt, a lifting crane and a press respectively. Parsing this statement it is easy to deduce that *Belt* (the feed belt) interfaces with *Table* (elevating table) and *Dbelt* (deposit belt) interfaces with *Crane* (the crane). Furthermore, each of these subsystems operates without direct interface to each other or with *Press*, but all components possibly interface with the robot. Working only from this model component it is not possible to deduce what actions the robot shares with each of the other individual components. This problem may be overcome by also parsing the component specifications to determine what components participate in which actions. The resultant representation is shown in Figure 3.5; the complexity of the components indicated by the size of the circles representing them. We have also animated this view to show model evolution along different paths.

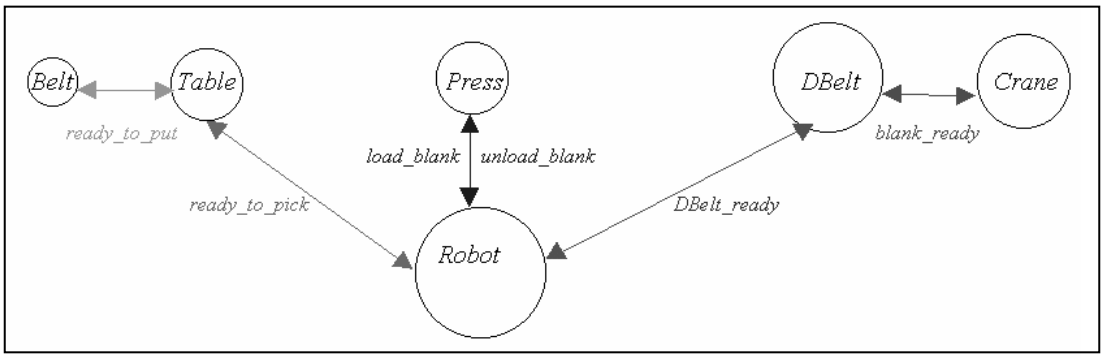

Figure 3.5: Component Interface view

As well as providing a useful high level representation in its own right, the visualisation of component interfaces also provides a useful navigation mechanism to support more detailed behaviour. Figure 3.6 shows a prototype navigation tool written in HTML using Holton's production cell example [27].

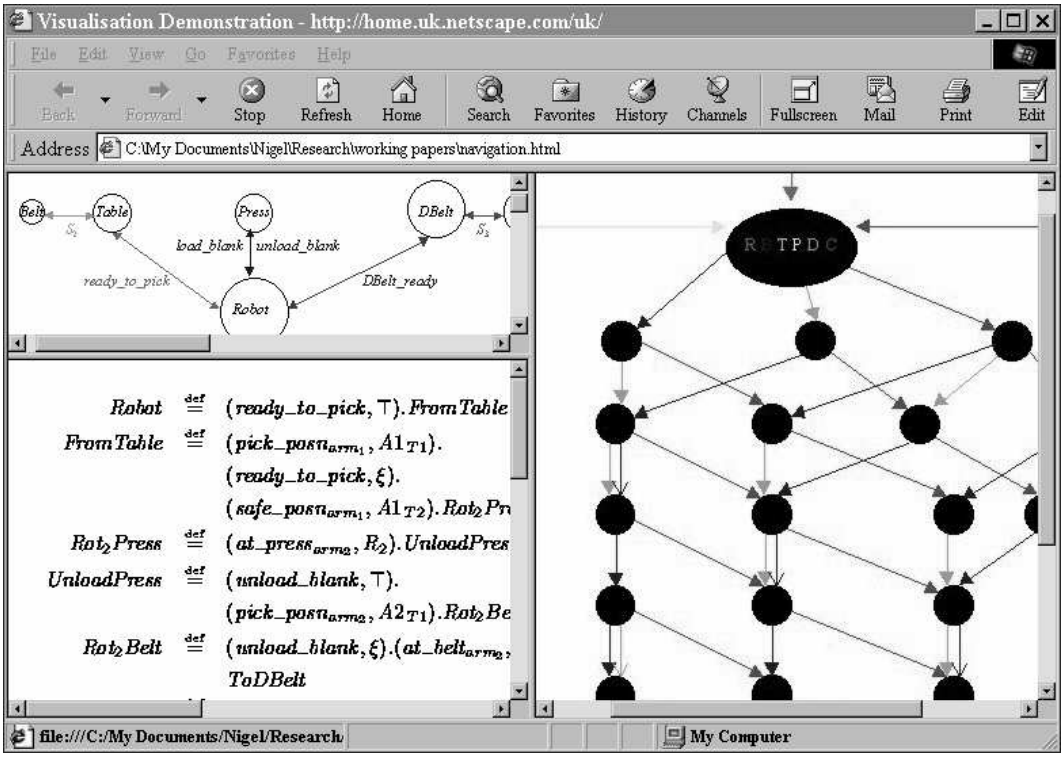

Figure 3.6: A prototype navigation tool

There are three frames, one contains the visual representation of the component interfaces, one contains the entire PEPA specification and the third contains a derivation graph view of a single component. The different representations are colour coordinated so that the user can see what component is in view by the colour it is displayed in. The interface diagram acts as a map; clicking on a component changes the PEPA script to the corresponding position. This prototype tool demonstrates how even very simple graphical representations can collectively give a far greater aid to comprehension.

4. Deriving UML from SPA models

Because stochastic process algebra, such as PEPA, are formally defined and have formal notions of equivalence it is possible to define transformations which automatically translate one specification to one or more others in a provably correct manner. However, the majority of the motivations for performing such transformations and some of the notions of equivalence used relate more to the state space of the solution than the behaviour of the model as related to the design [24]. It is therefore necessary for the designer to be able to have confidence that the altered model still relates to the design in an understandable way from a design perspective. An obvious solution to this problem might be to derive a UML specification directly from the altered SPA specification, the reverse of the process described by Pooley and King [37].

We consider three of the many modelling mechanisms used in UML, state charts, collaboration diagrams and sequence diagrams. State charts show the states and transitions of components. They record dependencies between the state of a component and its reaction to messages. Clearly there is a close relation to decomposed derivation graphs. Figure 3.8 shows UML state charts for the components of a resource use model specified in Figure 3.1.

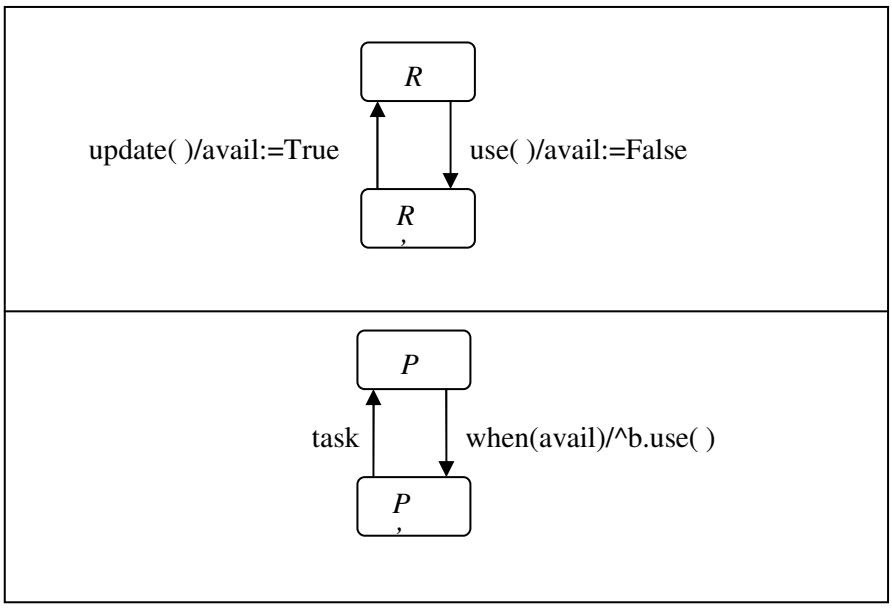

Figure 3.7: Statechart of a simple resource use model

The dependencies between the components over the shared *use* action are depicted by introducing an extra Boolean condition, *avail*. This condition provides the simple blocking mechanism on the shared action when it is not defined. The concurrent nature of the components is not explicitly stated and it is not clear whether the competition between instances of *P* is represented.

Collaboration diagrams record the links between components and their interactions. Clearly there is a close relation to the individual component interface view. Figure 3.9 shows a collaboration diagram for the model of resource use specified in Figure 3.1 and Figure 3.10 shows a collaboration diagram of Holton's model of a production workcell.

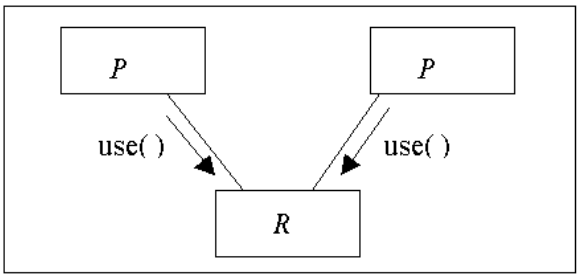

Figure 3.9: Collaboration diagram of a simple resource use model

It is worth comparing Figure 3.10 with the interface view of the same model shown in Figure 3.5. The interface view incorporates some additional features which are not available in UML, principally the use of colour, which facilitates comprehension, and the size of the circles used to indicate the complexity of components. The advantage of the UML diagram is that it is a standard which is more likely to be familiar to designers.

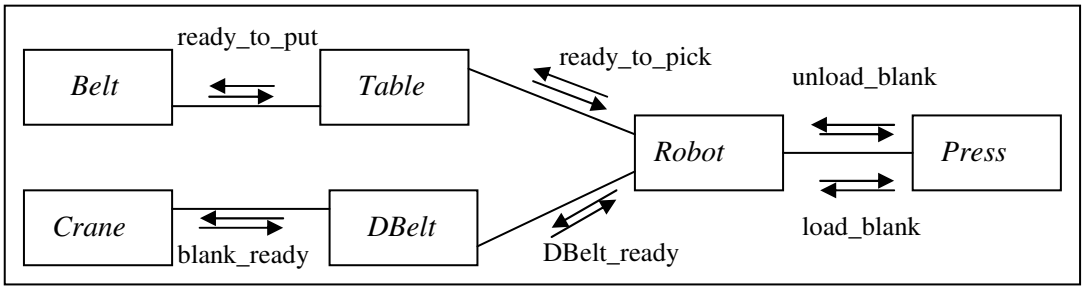

Figure 3.10: Collaboration diagram of Holton's workcell model

Sequence diagrams record necessary sequences of interactions between components and offer a similar functionality to animated component views. Figure 3.11 shows a sequence diagram of Holton's workcell model and Figure 3.12 shows a sequence diagram for the resource use model. Both diagrams show only partial evolution of the models.

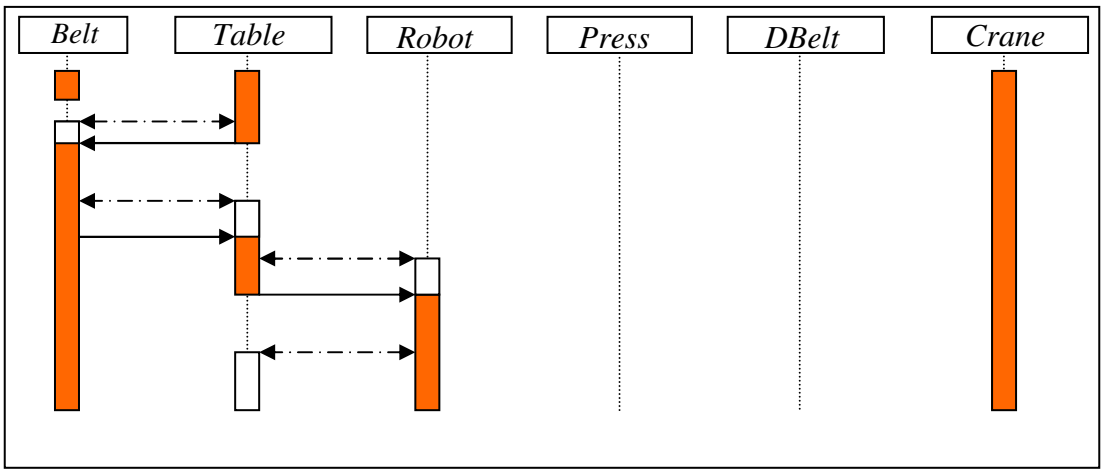

Figure 3.11: Sequence diagram of Holton's workcell model

The lines below each of the component names show time evolution, the boxes on those lines show when the component is activated. The colour in the box indicates that the component is

(or at least has the potential of) actively participating in actions, whereas a clear box indicates that the component may only be passively participating in actions. Where there is no box (only the time line) the component is in a blocked state.

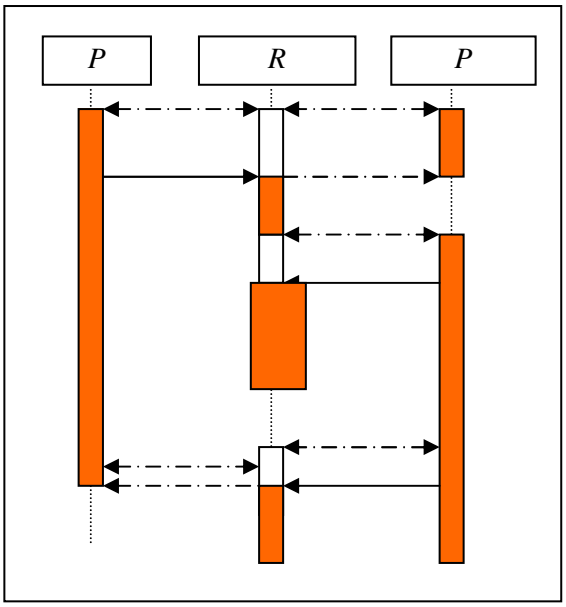

Figure 3.12: Sequence diagram of a resource use model

The difficulty in using sequence diagrams comes in adequately representing the synchronisation of shared actions. The approach we have used is to show an initialisation of a shared action as a two-way communication between the participating components (double headed arrow with dot-dashed line) and the end of a shared action as a one-way communication from the active to the passive partner. In the resource use model the is competition between the two instances of *P*, both of these are allowed to initialise their *use* actions, but obviously one will complete first causing the resource to become unavailable. This will block the other instance of *P*, so clearly there needs to be some communication from *R* to *P* to cease the use (this is shown as a single headed dot-dashed arrow).

The models for synchronisation in PEPA and the other stochastic process algebra have been subject for some debate during their development [25,7], the ultimate choice being made principally according to meaning in a stochastic model rather than any particular real world interpretation. Other models of synchronisation, such as synchronisation on points, as appear in message passing systems, rather than on shared actions, would have an easier interpretation in sequence diagrams.

4. Concluding remarks

This paper aims to present a position rather than a solution. Presented here are a set of ideas that head towards the goal of making performance models, in particular SPA models, more usable for the non-modelling expert. We are still a long way from our goal and much work remains to be done to develop the notions we have presented here, however we are creating a direction that is reducing the concept gap between modellers and designers. In particular we are aiming at presenting models and model solutions from a component perspective that the designer can identify with, rather than a global state space view so often favoured by modellers.

One area where progress has been slow is in presenting the stochastic nature of models. Software engineers have rarely had an education that includes much probability theory, hence understanding the different behaviours that arise from using different probability distributions is difficult. A clear example of this is the over reliance on averages as performance measure, but when the variance for that property is large the average behaviour is almost never actually observed. Another case in point is the time to reach a steady state. In reality a system may take days or weeks to reach a steady state, by which time the traffic profile may have altered considerably. The performance modeller can, with a little more difficulty, derive variance, tail distributions and transient solutions, but these must be understood by the designers and managers to be truly useful. Clearly much remains to be done.

Acknowledgements

This paper is the result of a tutorial presented at the Second ACM International Workshop on Software Performance (WOSP 2000) and brings together several years of work, much of which has been published already. Although there are many people who have contributed, knowingly or otherwise, to the position presented here the following deserve particular mention: Jane Hillston and Stephen Gilmore from the University of Edinburgh, Peter King and Rob Pooley from Heriot Watt University, Neil Davies of Degree 2 Innovations Limited and Malcolm Munro and Claire Knight of the University of Durham. The Dr PEPA tool was developed by Jim Newton as his final year undergraduate project at the University of Durham.

Dr Thomas and Dr Bradley are supported by the EPSRC funded QWiD project (reference GR/N01491).

Bibliography

[1] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis, *Modelling with Generalized Stochastic Petri Nets*, Wiley, 1995.

[2] M. Bernardo, Theory and Application of Extended Markovian Process Algebra, PhD Thesis, University of Bologna, 1999.

[3] M. Bernardo, N. Busi and M. Ribaudo, Integrating Two Towers and GreatSPN, in: *Proceedings of 8th International Workshop on Process Algebra and Performance Modelling*, ICALP Workshops 2000, Proceedings in Informatics 8, pp. 551-564, Carleton Scientific Publishers, 2000.

[4] M. Bernardo and R. Gorrieri, A Tutorial on EMPA: A Theory of Concurrent Processes with Nondeterminism, Priorities, Probabilities and Time, *Theoretical Computer Science*, 202(1-2), pp. 1-54, 1998.

[5] M. Bernardo, L. Donatiello and R. Gorrieri, Describing Queueing Systems with MPA, Technical Report UBLCS-94-11, University of Bologna, May, 1994.

[6] H.C. Bohnenkamp and B.R. Haverkort, Stochastic Event Structures for the Decomposition of Stochastic Process Algebra Models, in: *Proceedings of 7th International Workshop on Process Algebra and Performance Modelling*, pp. 25-39, Zaragoza, 1999.

[7] J.T. Bradley and N.J. Davies, Reliable Performance Modelling with Appropriate Synchronisations, in: *Proceedings of 7th International Workshop on Process Algebra and Performance Modelling*, pp. 25-39, Zaragoza, 1999.

[8] J.T. Bradley and N.J. Davies, Measuring Improved Reliability in Stochastic Systems, in: J.T. Bradley and N.J. Davies (eds.), *Proceedings of the Fifteenth UK Performance Engineering Workshop*, pp. 121-130, University of Bristol, July 1999.

[9] J.T. Bradley and N.J. Davies, A Matrix-based Method for Analysing Stochastic Process Algebras, in: *Proceedings of 8th International Workshop on Process Algebra and*

Performance Modelling, ICALP Workshops 2000, Proceedings in Informatics 8, Carleton Scientific Publishers, 2000.

[10] J.T. Bradley and Nigel Thomas, Constructing a partial order for performance measures, in: N. Thomas and J. Bradley (eds.) *Proceedings of the Sixteenth UK Performance Engineering Workshop*, University of Durham, July 2000.

[11] G. Clark, S. Gilmore, J. Hillston, and N. Thomas, Experiences with the PEPA performance modelling tools, *IEE Proceedings - Software*, 146(1), pp.11-19, 1999.

[12] R. Cleaveland, P. M. Lewis, S. A. Smolka and O. Sokolsky, The Concurrency Factory: A Development Environment for Concurrent Systems, in: R. Alur and T. Henzinger, (eds.), *Proceedings of Computer-Aided Verification*, volume 1102 of Lecture Notes in Computer Science, Springer-Verlag, pp. 398-401, 1996.

[13] D.R. Cox and H.D. Miller, *The Theory of Stochastic Processes*, Methuen, 1965.

[14] S. Donatelli, J. Hillston, and M. Ribaudo, A comparison of Performance Evaluation Process Algebra and Generalized Stochastic Petri Nets, in: *Proceedings of 6th International Workshop on Petri Nets and Performance Models*, Durham, North Carolina, 1995.

[15] S. Donatelli, J. Hillston, and M. Ribaudo, A Comparison of Performance Evaluation Process Algebra and Generalized Stochastic Petri Nets, Technical Report ECS-CSG-6-94, Department of Computer Science, The University of Edinburgh, 1994.

[16] S. Gilmore and J. Hillston, Performance modelling in PEPA with higher order functions, in: N. Thomas and J. Bradley (eds.) *Proceedings of the Sixteenth UK Performance Engineering Workshop*, University of Durham, July 2000.

[17] 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*, number 794 in Lecture Notes in Computer Science, pages 353-368, Vienna, May 1994. Springer-Verlag.

[18] S. Gilmore, J. Hillston, and D.R.W. Holton, From SPA models to programs, in: M. Ribaudo (ed.), *Proceedings of the Fourth International Workshop on Process Algebra and Performance Modelling*, pp. 179-198, Università di Torino, July 1996.

[19] N. Götz, U. Herzog and M. Rettelbach, TIPP - a language for timed processes and performance evaluation, in: J. Hillston and F. Moller eds., *Proceedings of the First International Workshop on Process Algebra and Performance Modelling*, University of Edinburgh, UK, May, 1993.

[20] D. Harel, Statecharts: A visual formalism for complex systems, *Science of Computer Programming*, 8, pp. 231-274, 1987.

[21] D. Harel and M. Politi, *Modelling Reactive Systems with State charts*, McGrawHill, 1998.

[22] B.R. Haverkort, Are stochastic process algebras good for performance and dependability evaluation, in: ICALP Workshops 2000, Proceedings in Informatics 8, pp. 501-510, Carleton Scientific Publishers, 2000.

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

[24] J. Hillston, Exploiting structure in solution: Decomposing composed models, in: C. Priami (ed.), *Proceedings of the Sixth International Workshop on Process Algebra and Performance Modelling*, Università degli studi di Verona, September 1998.

[25] J. Hillston, The Nature of Synchronisaton, in: U. Herzog and M. Rettelbach (eds.), *Proceedings of the Second International Workshop on Process Algebra and Performance Modelling*, Erlangen, 1994.

[26] C.A.R. Hoare, *Communicating Sequential Processes*, Prentice Hall, 1985.

[27] D.R.W. Holton, A PEPA specification of an industrial production cell, The Computer Journal, 38(7), pp. 542-551, December 1995.

[28] C. Kabajunga and R.J. Pooley, Simulating UML sequence diagrams, in R.J. Pooley and N. Thomas, editors, UK PEW 1998, pages 198--207. UK Performance Engineering Workshop, July 1998.

[29] P. Kähkipuro, UML based performance modelling framework for object-oriented distributed systems, in: *«UML» '99 - The Unified Modelling Language: Beyond the Standard*, pp. 356-371, October 1999.
[30] P. Kähkipuro, Performance Modeling Framework for CORBA Based Distributed Systems, PhD Thesis, Department of Computer Science, University of Helsinki, 2000.
[31] C. Knight and M. Munro, Visualising software - a key research area, in: *Proceedings of IEEE International Conference on Software Maintenance*, Oxford, 1999.
[32] C. Knight, Visualisation for program comprehension: information and issues, Department of Computer Science Technical Report, University of Durham, 1998.
[33] J. Maggott and Z. Huzar, New Semantics for Markovian Statecharts, in: N. Thomas and J. Bradley (eds.) *Proceedings of the Sixteenth UK Performance Engineering Workshop*, University of Durham, July 2000.
[34] R. Milner, Communication and concurrency, Prentice Hall, 1989.
[35] P. Mitton and D.R.W. Holton, Performability modelling using UML statecharts, in: N. Thomas and J. Bradley (eds.) *Proceedings of the Sixteenth UK Performance Engineering Workshop*, University of Durham, July 2000.
[36] P. Pettersson and K.G. Larsen, UPPAAL2k, *Bulletin of the European Association for Theoretical Computer Science*, volume 70, pp. 40-44, 2000.
[37] R.J. Pooley and P.J.B. King, Using UML to derive stochastic process algebra models, in: J.T. Bradley and N.J. Davies (eds.), *Proceedings of the Fifteenth UK Performance Engineering Workshop*, pp. 23-33, University of Bristol, July 1999.
[38] R.J. Pooley and P.J.B. King, The Unified Modelling Language and performance engineering, *IEE Proceedings - Software*, 146(1), pp. 2-10, February 1999.
[39] O. Rana, M. Shields and A. Jones, Analysing Java execution semantics using stochastic Petri Nets, in: N. Thomas and J. Bradley (eds.) *Proceedings of the Sixteenth UK Performance Engineering Workshop*, University of Durham, July 2000.
[40] M. Ribaudo. Stochastic Petri net semantics for stochastic process algebras. In *Proc. 6th International Workshop on Petri Nets and Performance Models*, Durham NC, 1995.
[41] N. Thomas and J. Hillston, Markovian Queueing Systems modelled with PEPA, in: R.J. Pooley and N. Thomas, eds., *Proceedings of 14th UK Performance Engineering Workshop*, Edinburgh, July, 1998.
[42] N. Thomas and J. Hillston. Using Markovian process algebra to specify interactions in queueing systems. Technical Report ECS-LFCS-97-373, Laboratory for Foundations of Computer Science, The University of Edinburgh, 1997.
[43] N. Thomas, M. Munro, P.J.B. King and R.J. Pooley, Visualisation for model comprehension, in: N. Thomas and J. Bradley (eds.) *Proceedings of the Sixteenth UK Performance Engineering Workshop*, University of Durham, July 2000.
[44] N. Thomas, M. Munro, P.J.B. King and R.J. Pooley, Visualisation of Stochastic Process Algebra Models, in: *Proceedings of Second ACM International Workshop on Software Performance*, Ottawa, September 2000.
[45] P.J. Young and M. Munro, Visualising software in virtual reality, in*: Proceedings of IEEE 7th International Workshop on Program Comprehension*, pp. 19-26, May 1998.
[46] P.J. Young and M. Munro, A new view of call graphs for visualising call structures, Department of Computer Science Technical Report, University of Durham, April 1997.