Visual representation of stochastic process algebra models

Nigel Thomas[†] Malcolm Munro[†]
{nigel.thomas|malcolm.munro}@durham.ac.uk

Peter King[‡] Rob Pooley[‡]
{pjbk|rjp}@cee.hw.ac.uk

ABSTRACT
Performance models are of increasing interest to professionals who do not have a background in mathematical analysis, it is important to provide additional mechanisms by developers may improve their confidence in the models they evaluate accurately reflect the systems they are studying. In this paper we discuss these issues, suggesting some basic techniques from program comprehension that be applied to models in a stochastic process algebra

Keywords
Visualisation, process algebra, model comprehension.

1. INTRODUCTION
In traditional software engineering visualisation techniques are established aids to understanding complex software systems. Performance modelling tools supporting graphical representations, using Petri nets and queueing theory, can aid model comprehension for practitioners used to those formalisms, but offer little to the uninitiated. 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), are an extremely good modelling paradigm for application in a wide variety of different domains. It has been successfully demonstrated that performance models can be derived from design specifications in UML [4] and that SPA models can be related directly to code. These developments have led to performance models being studied by software engineers with little or no experience of stochastic modelling. Some earlier work with classical process algebra has investigated the use of visual techniques for model construction, e.g. CCS [6]; our objectives are to aid post developmental understanding, although some of the approaches are similar.

A typical performance oriented development 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 [4], 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. It is clear that much is being done to promote confidence in performance models and measures, but these techniques only address specification, solution and application; there have not, as yet, been any attempts to relate the original model, or the altered model, to the design.

The mechanisms by which a specification is altered to derive a new model that has desirable properties, e.g. simplicity and solvability, are varied. Because stochastic process algebra, such as PEPA [1], 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. 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. A solution to this problem might be to derive a UML specification directly from a SPA model, unfortunately the known set of UML mappings to SPA models is limited. If the reverse process were to be possible it would be necessary to ensure that the model manipulation does not perform transformations that derive models which lay outside this restricted subset.

WOSP 2000, Ontario, Canada

2. VISUAL REPRESENTATIONS OF PEPA DERIVATION GRAPHS
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 to the modeller by the PEPA Workbench tools [1], but rather is an important mechanism used to study the PEPA model and derive a numerical solution. There is a clear relation between the properties of the graph and the properties of the PEPA model, for instance if the graph is cyclic, then so is the PEPA model and vice versa. The derivation graph is a directed graph where the nodes represent the evolution of the cooperating

PEPA agents and the arcs represent the activities that are carried out. If a steady state solution to the model exists then the graph is strongly connected and cyclic, with every node reachable from every other. 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. The graph may be enhanced using colour to show what agents are participating in the actions and are subsequently evolved. Given a large state space the size of the full derivation graph in this form quickly becomes unmanageable. Several possibilities exist for handling problems of scale in presentation: elimination, aggregation, decomposition, highlighting, layering, abstraction and windowing.

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 an activity oriented view is a simple matter of exchanging nodes and, alternatively we could use call stack representations which have been used successfully in program visualisation [5].

3. REPRESENTING COMPONENT INTERFACES

In the previous section we discussed 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. Consider for example this statement taken from a production cell model [2]:

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

$$S_1 = \{ ready\_to\_pick, unload\_blank, DBelt\_ready, load\_blank \}$$

$$S_2 = \{ ready\_to\_put \} \qquad S_3 = \{ blank\_ready \}$$

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. 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. We have developed a prototype navigation tool written in HTML, incorporating 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 an aid to comprehension.

4. CONCLUSIONS AND FURTHER WORK

The focus of this paper is quite different from most previous work with stochastic process algebra in that it is not mathematically challenging neither does it enable larger or more complex models to be studied. However, it is crucial that performance analysis with stochastic process algebra is made more accessible if it is to gain general acceptance in software engineering practice. In this paper we have shown how some simple graphical representations can aid the understanding of model specifications and promote confidence in those models amongst system designers. To date all the graphs we have used have been prepared manually, this is adequate for the purposes of demonstration with moderately sized models, but for practical purposes it will be necessary to automate this process further. To this end we aim to produce a set of visualisation tools to further develop the ideas we have presented in this paper. Other techniques to be investigated further include the use of symbolism and shape to identify different object types, size, texture, tone and hue to present different object properties and layout and lines to depict relational properties. Another possible direction is to exploit concepts from the application domain to provide more meaningful graphical representations. We also hope to be able to depict the stochastic nature of these models.

5. REFERENCES

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

[2] D.R.W. Holton, A PEPA specification of an industrial production cell, The Computer Journal, 38(7), 1995.

[3] C. Knight and M. Munro, Visualising software - a key research area, *Proceedings of IEEE International Conference on Software Maintenance*, 1999.

[4] R.J. Pooley and P.J.B. King, Using UML to derive stochastic process algebra models, *Proceedings of the Fifteenth UK Performance Engineering Workshop*, 1999.

[5] P.J. Young and M. Munro, Visualising software in virtual reality, *Proceedings of IEEE 7th International Workshop on Program Comprehension*, 1998.

[6] R. Milner, Communication and concurrency, Prentice-Hall, 1989.

† Research Institute in Software Evolution, University of Durham, Durham, UK

‡ Department of Computing and Electrical Engineering, Heriot Watt University, Edinburgh, UK