A Unified Tool for Performance Modelling and Prediction

Stephen Gilmore and Leïla Kloul*

Laboratory for Foundations of Computer Science, The University of Edinburgh, Edinburgh, Scotland, EH9 3JZ

Abstract. We describe a novel performability modelling approach which facilitates the efficient solution of performance models extracted from high-level descriptions of systems. The notation which we use for our high-level designs is the UML graphical modelling language. The technology which provides the efficient representation capability for the underlying performance model is the MTBDD-based PRISM probabilistic model checker. The UML models are compiled through an intermediate language, the stochastic process algebra PEPA, before translation into MTBDDs for solution. We illustrate our approach on a real-world analysis problem from the domain of mobile telephony.

1 Introduction

Distributed, mobile and global computing environments provide robust development challenges to practising software system developers. Working with rapidly-changing implementation technology means that developers often must spend some of their development time finding and correcting errors in the software libraries and APIs which they use. Fortifying this difficulty is the arduous terrain of dynamic distributed systems where the difficulty of replaying a communication sequence which led to a system fault confounds the process of detecting and correcting implementation errors.

In this setting, application developers rarely wish to expend the investment of time which would be needed to build and analyse a performance model of the system which they are developing. The concepts and the modelling languages of performance analysis are relatively unfamiliar to software developers and when already faced with a generous range of other difficulties in the development process, early predictive performance analysis can easily be overlooked.

However, this is an imprudent practice. If performance design flaws are found early in the development process then they can be corrected at a relatively low cost. In contrast, if they are found after the development process is long underway then they may be very expensive or even unrealistic to repair. If they are then subsequently ignored, such problems will lead to unreliability of the high-end

* On leave from PRiSM, Université de Versailles, 45, av. des Etats-Unis, 78000 Versailles, France

consumer electronic devices which are increasingly used in safety-critical contexts thoughout society.

To combat these difficulties, a performance modelling methodology which is designed for effective development of such global, mobile or high-end distributed systems should provide at least the following two features: a convenient high-level modelling notation for expressing performance models; and efficient solution methods for realistic models of complex systems. Unfortunately these two requirements are often at variance. In order to access state-of-the-art solution methods one usually additionally needs to master sophisticated representation and analysis methods which sometimes are inconvenient or troublesome to use. Conversely, high-level modelling platforms devote much of their efforts to providing reliable graphical editors and supporting document and IDE infrastructure and this can come at the expense of equipping them with complementary analysis tools.

We provide a structured performance engineering platform for this problem domain by connecting a *specification environment* (SENV) and a *verification environment* (VENV) so that each may communicate with the other. The SENV and VENV are connected by a bridge which consists of two categories of software tools. These are:

- *extractors* which translate designs from the SENV into inputs for the VENV, omitting any aspects of the design which are not relevant for the verification task at hand; and
- *reflectors* which convert the results from the analysis performed by the VENV back into a form which can be processed and displayed by the SENV.

A series of extractors can be chained together to provide a path from one specification formalism to another. Similarly, reflectors can be chained together in order that the results of one analysis process may be presented back in the format of another. A process of *extractor/reflector chaining* is used here to connect a specification environment to multiple verification environments. We use the ArgoUML design environment [1] as our SENV and the PEPA Workbench [2] and the PRISM probabilistic symbolic model checker [3] both play the role of VENVs. ArgoUML provides the Unified Modelling Language (UML) [4] as its modelling language. The PEPA Workbench and PRISM both support the PEPA stochastic process algebra [5]. PRISM additionally supports a state-based language based on the Reactive Modules formalism of Alur and Henzinger [6].

Structure of this paper: In the next section we describe some of the background to this work, providing a summary of UML, PEPA and PRISM modelling. In Section 3 we describe the software architecture of the system, as an integrated set of components. We give details of the implementation, providing an explanation of how the components work together and calling attention to cases where some care has been needed. In Section 4 we present our case study, showing the approach applied to a realistic example. In Section 5 we survey related work. Conclusions are presented in Section 6.

2 Background

The Unified Modelling Language (UML) is an effective diagrammatic notation used to capture high-level designs of systems, especially object-oriented software systems. A UML *model* is represented by a collection of diagrams describing parts of the system from different points of view; there are seven main diagram types. For example, there will typically be a *static structure diagram* (or *class diagram*) describing the classes and interfaces in the system and their static relationships (inheritance, dependency, etc.). State diagrams, a variant of Harel state charts, can be used to record dynamic behaviour. Interaction diagrams, such as sequence diagrams, are used to illustrate the way objects of different classes interact in a particular scenario. As usual we expect that the UML modeller will make a number of diagrams of different kinds. Our analysis is based on state and collaboration diagrams.

We have introduced performance information in the state diagrams such that each transition in these diagrams is labelled with a pair 'a / rate(r)' where a is the action type executed and r is an exponentially distributed rate associated with this action. We often simplify the representation of the transition labels in order to save space writing this as 'a / r'. A customer arrival causes a change in the state of a queue so this would be one example of an action type which we might use. Concretely, *arrive*/rate(λ) and *serve*/rate(μ) would be suitable arc adornments for a state diagram for a queue (abbreviated to *arrive*/λ and *serve*/μ).

In Performance Evaluation Process Algebra (PEPA) [5], a system is viewed as a set of *components* which carry out *activities* either individually or in cooperation with other components. Activities which are private to the component in which they occur are represented by the distinguished action type, τ. Each activity is characterized by an *action type* and a duration which is exponentially distributed. This is written as a pair such as (α, r) where α is the action type and r is the *activity rate*. This parameter may be any positive real number, or may be unspecified. We use the distinguished symbol $\top$ to indicate that the rate is not specified by this component. This component is said to be *passive* with respect to this action type and the rate of the shared activity is defined by another component.

PEPA provides a set of combinators which allow expressions to be built which define the behaviour of components via the activities that they engage in. These combinators are presented below.

Prefix $(\alpha, r).P$: Prefix is the basic mechanism by which the behaviours of components are constructed. This combinator implies that after the component has carried out activity (α, r), it behaves as component P.

Choice $P_1 + P_2$: This combinator represents a competition between components. The system may behave either as component P_1 or as P_2. All current activities of the two components are enabled. The first activity to complete distinguishes one of these components and the other is then discarded.

Cooperation $P_1 \underset{L}{\bowtie} P_2$: This describes the synchronization of components P_1 and P_2 over the activities in the cooperation set L. The components may proceed independently with activities whose types do not belong to this set. A particular case of the cooperation is when $L = \emptyset$. In this case, components proceed with all activities independently. The notation $P_1 \parallel P_2$ is used as a shorthand for $P_1 \bowtie_{\emptyset} P_2$. In a cooperation, the rate of a shared activity is defined as the rate of the slowest component.

Hiding: P/L This component behaves like P except that any activities of types within the set L are *hidden*, i.e. such an activity exhibits the unknown type τ and the activity can be regarded as an internal delay by the component. Such an activity cannot be carried out in cooperation with any other component: the original action type of a hidden activity is no longer externally accessible, to an observer or to another component; the duration is unaffected.

Constant: $A \stackrel{def}{=} P$ Constants are components whose meaning is given by a defining equation: $A \stackrel{def}{=} P$ gives the constant A the behaviour of the component P. This is how we assign names to components (behaviours). An explicit recursion operator is not provided but components of infinite behaviour may be readily described using sets of mutually recursive defining equations.

The transition system underlying the PEPA model gives the continuous time Markov process represented by the model. The generation of this process is based on the derivation graph of the model in which syntactic terms form the nodes, and arcs represent the possible transitions between them. This derivation graph describes the possible behaviour of any PEPA component and provides a useful way to reason about a model. The use of the derivation graph is analogous to the use of the reachability graph in stochastic extensions of Petri nets such as GSPN [5].

Specifications are constructed in PRISM by defining a collection of reactive modules which synchronise on shared activities. The state of each module is determined by a set of local variables. The models which we work with here are all obtained by compiling an input PEPA model. All of these have associated constants which enumerate the states of the module and use a single local variable to record the current state. We developed the PEPA-to-PRISM compiler to allow us to analyse our PEPA models with PRISM. Our compiler has been incorporated into the latest release of PRISM (version 1.3) [7].

The behaviour of a reactive module is encoded by a list of guarded transitions which name the activity performed and specify assignments to the local variables which are to be carried out if the activity is performed. The PRISM model-checker accepts descriptions of discrete-time Markov chains (DTMCs), continuous-time Markov chains (CTMCs) and Markov decision processes (MDPs).

CSL model-checking of CTMCs allows the user to check performability properties which combine probabilities, behaviour and time, such as "the probability that a hand off call will be dropped within 100 time units is less than 0.1."

We use PRISM to solve PEPA models for their stationary probability distribution. The first step of this process is generating the full state space of the system. This is compactly stored by PRISM as a multi-terminal binary decision diagram (MTBDD). The CUDD package [8] is used as a library, providing MTBDD data structures and algorithms to PRISM.

3 The Software Architecture

Ours is a component-based software architecture in which we link substantial software tools with lightweight connectors called extractors and reflectors. This promotes significant code re-use and allows for clean interfaces between systems using formal description languages such as PEPA and PRISM's reactive modules.

A performance modeller using our tool will design a system using a UML modelling environment such as ArgoUML. UML software tools produce XML-based model interchange files called XMI files. The XMI file of the model is used by the PEPA Workbench to extract a PEPA model. The resulting PEPA file is then submitted to the model checker PRISM. PRISM produces a steady state probability vector. This is then reflected by the PRISM reflector as a PEPA steady state probability vector. Using some information in the XMI file produced initially by the UML tool, the PEPA workbench reflects these results as an XMI document.

Finally the modeller can then visualize the performance measures obtained since this last reflector of the chaining process consists in adding these new information into the user UML model.

The start of our UML performance model analysis process is an `.xmi` or `.zargo` file obtained from ArgoUML or a similar UML tool. The PEPA Workbench contains as components the PEPA Extractor and the PEPA Reflector which convert between UML models and other kinds. The PEPA Extractor is first used to process the file containing the UML model and extracts a PEPA model from this. This PEPA model is then compiled using the PEPA-to-PRISM compiler and solved by PRISM. The PRISM Reflector assumes that the PEPA Extractor and compiler have already been run. The former has extracted a `.pepa` file from an `.xmi` or `.zargo` file. The latter has extracted a `.sm` file from the `.pepa` file and has written a log file (`.log`) mapping PEPA local state identifiers onto the numeric constants used in the reactive modules notation. The output from the PRISM tool onto the standard output stream has been captured and saved in a `.pres` (PRISM results) file. The PRISM Extractor reads the `.log` file and the `.pres` file and writes an `.xml` file in the same format as the PEPA Workbench. PEPA Workbench results files can be read by the PEPA Reflector. We use the PEPA Reflector next to merge the results with the orginial input UML model to produce a modified `.xmi` or `.zargo` file which includes the results of the performance analysis. This file can be loaded into the ArgoUML tool to present the results back to the UML performance modeller.

Our extractor and reflector software tools are implemented in the programming languages Java and Standard ML [9]. The Java components handle the processing of the zipped archives (in `.zargo` format) of UML models produced by the ArgoUML tool and the parsing of the XMI documents containing the UML models (in `.xmi` format). The DOM parser from the Java 1.4 `javax.xml` package is used to parse the XMI files. Much of the more complex processing is implemented in the Standard ML language.

We now present a case study which demonstrates the use of these tools.

4 Case study: hierarchical cellular network

The hierarchical cellular network consists of two tiers of cells, a level of macrocells overlying a level of microcells. In this study, we consider the Manhattan model [10] where the reuse pattern is based on a five squared microcell cluster, a central cell surrounded by four peripheral cells (Fig.1). This model takes its name from the city of Manhattan which consists of square-blocks, representing buildings, with streets in between them.

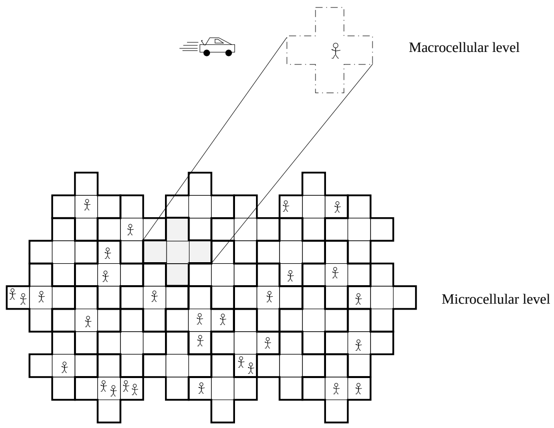

Fig. 1. Reuse pattern of Manhattan model

As in this model each microcell has four neighbouring cells, we consider a microcell cluster model composed of a central microcell surrounded by four peripheral cells. We consider a FCA scheme (Fixed Channel Allocation [11]), where S channels are distributed among the different cells. Let c_j, $j = 1 \dots 5$ be the capacity of microcell j and c_0 the capacity of the macrocell.

Considering a homogeneous system in statistical equilibrium, any cluster of microcells overlaid by a macrocell has statistically the same behaviour as any other cluster of microcells overlayed by a macrocell. We use this observation to decouple a cluster from the rest of the system. That is, we can analyze the overall system by focusing on a given cluster under the condition that the neighbouring clusters exhibit their typical random behaviour independently.

We consider two types of customers inside the cluster, the new calls and the handover requests (ongoing calls). External arrivals to the cluster consist of the handover requests coming from other clusters and the new calls initiated in that cluster. We assume that the handover requests coming from other clusters may occur only in the macrocell or the peripheral microcells. We consider that these arrivals may never occur in the central microcell.

In this study, new calls can be assigned only to the microcell level. Moreover, we consider a hierarchical cellular network using an overflow strategy but without reversible capability, except for the external arrivals to the macrocell. A request, either a new call or a handover request, initiated at the microcell level is served in its originating microcell if a channel is available. Otherwise, according to the overflow strategy, the request is overflowed to the upper level and is satisfied if a channel is free at this level. In the case where all channels are busy at both levels, the request is dropped (in the case of a handover) or blocked (in the case of a new call).

This system is studied under the usual Markovian assumptions. New call and handover request arrivals follow a Poisson process. We assume that the average new call arrival rates and the handover arrival rates are the same for all cells in the network. The session duration which represents the duration of a communication is modelled by a service time which is exponentially distributed with parameter μ. The amount of time that a user remains within a coverage cell of a given base station, called *dwell-time*, is assumed to be exponentially distributed with parameter α.

In the next section we present the UML model corresponding to this system.

4.1 The UML Model

In the model, the external arrival process is represented by the event *in* by the cells. The arrival rate is assumed to be λ_1 in the macrocell, λ_2 in the peripheral microcells and λ_3 in the central microcell.

Because of the different types of cells (macrocell, peripheral microcell, central microcell) and the topology of the network, we make a distinction between handover requests generated by the cluster itself (Fig. 2). This distinction is based

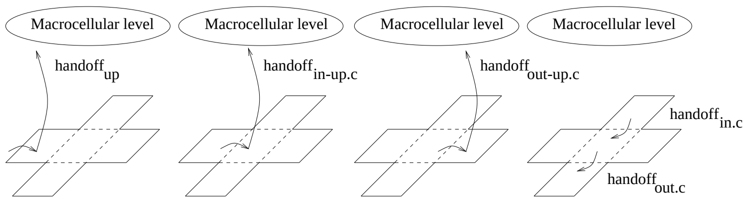

Fig. 2. The handoff requests graph

on the cell type this request originated from and the cell type satisfying this request, which means the cell where the ongoing call has to be transferred to. Thus, the arrival process of these customers is represented by the event *handoff* indexed by the type of handover request as follows:

- $handoff_{down}$ represents the transfer of a call from the macrocell to a microcell. This call is a handover request coming from outside the cluster to the macrocell and because all its channels are busy, it has to be transferred to a microcell,

- *handoff* $_{up}$ represents the transfer of a call from a microcell to the macrocell. This call may be either a new call or a handover request coming from outside the cluster to the microcell and because all channels in the microcell are busy, it has to be transferred to the macrocell. The rate associated with this event is the external arrival rate to the microcell,
- *handoff* $_{in.c}$ is the event which triggers the transfer of an ongoing call from one of the peripheral microcells to the central microcell,
- in contrast, *handoff* $_{out.c}$ represents the transfer of an ongoing call from the central microcell to one of the four peripheral microcells,
- *handoff* $_{in-up.c}$ represents the case where an ongoing call coming from a peripheral microcell and entering the central microcell, is then transferred to the macro cell because all channels of the central microcell are busy,
- *handoff* $_{out-up.c}$ models the arrival of an ongoing call from the central microcell to a peripheral microcell and because all channels of this cell are busy, the handover call is overflowed to the macro cell.

As the process behind the four last *handoff* events is the same, the corresponding rate is also the same and it is denoted by α (representing the mean dwell-time in a microcell). In all cells, the service process is represented by event *service*. As the service rate in each cell is assumed to be μ, when there are i, $1 \leq i \leq c_k$, customers in a cell, the event *service* is of rate $i\mu$.

In the following, we present the state diagrams of the different components of our network and the collaboration diagram showing the interactions between these components.

The State Diagrams We denote by *macro* the macrocell overlying the cluster of microcells. *microc* and *microj*, denote the central microcell and a peripheral microcell respectively. For the sake of readability of the different state diagrams, we limit the total number of channels to $S = 18$ and these channels are fairly shared by the different cells: $c_j = 3$, $j = 0 \ldots 5$.

The state diagram of *macro* is described in Fig. 3 where $macro_i$, $i = 0 \ldots 3$, is the state where i channels are busy.

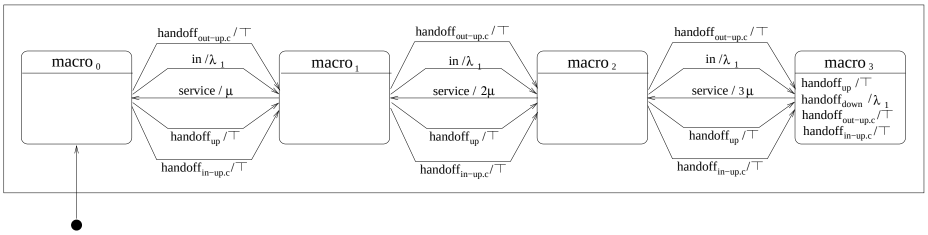

Fig. 3. The State diagram of *macro*

A transition from state $macro_i$ to state $macro_{i+1}$ denotes the arrival of either a call from outside the cluster (*in*) or a call from one of the microcells with *handoff* $_{up}$, *handoff* $_{in-up.c}$ or *handoff* $_{out-up.c}$. A natural termination of a call is represented by a transition from state $macro_i$ to state $macro_{i-1}$ with *service*.

When all channels are busy (state $macro_3$), if a handover call arrives from the microcells, the call is dropped and thus lost. Similarly, if a external handover call arrives when all channels are busy, the call is blocked and lost.

The state diagram of *microj* is described in Fig. 4 where $microj_k$, $k = 0 \ldots 3$, is the state where k channels are busy.

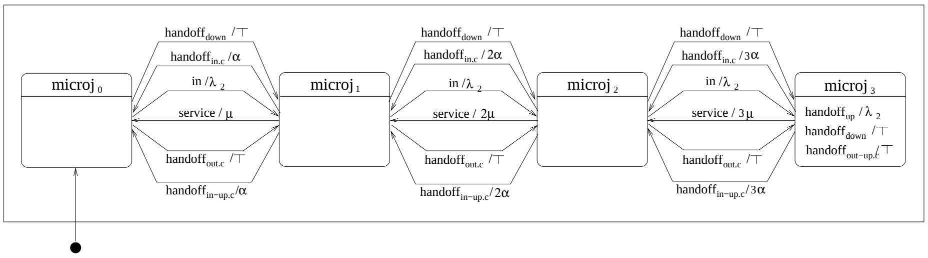

Fig. 4. The State diagram of a peripheral microcell *microj*

A transition from state $microj_k$ to state $microj_{k+1}$ denotes the arrival of either a new call (*in*), an ongoing call from the central microcell (*handoff* $_{out.c}$) or from the macrocell (*handoff* $_{down}$). The departure of a call from a peripheral microcell to the central microcell (*handoff* $_{in.c}$) or to the macrocell via the central (*handoff* $_{in-up.c}$) is depicted in the diagram by a transition from $microj_k$ to state $microj_{k-1}$. A similar transition with *service* models a natural termination of a call.

As for the *macro*, all arrivals when all channels of a peripheral cell are full (state $microj_3$) are lost.

The state diagram of the central microcell *microc* is described in Fig. 5 where $microc_i$, $i = 0 \ldots 3$, is the state where i channels are busy.

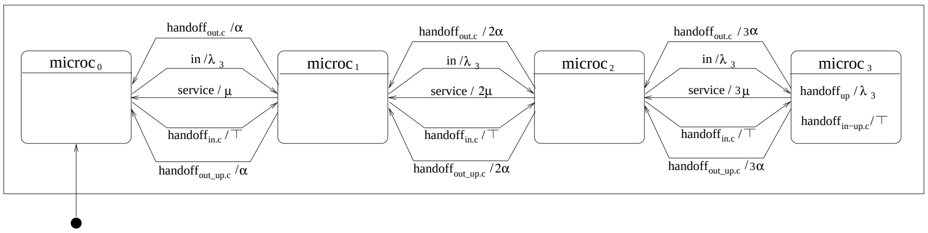

Fig. 5. The State diagram of the central microcell *microc*

In this diagram, a transition from state $microc_i$ to state $microc_{i+1}$ denotes the arrival of either a new call (*in*) or an ongoing call from a peripheral microcell ($handoff_{in.c}$). A call leaving the central microcell for a peripheral microcell ($handoff_{out.c}$) or for the macrocell via a peripheral one ($handoff_{out-up.c}$) is depicted by a transition from $microc_i$ to state $microc_{i-1}$.

The Collaboration Diagram In the network, the peripheral microcells will behave independently, but will synchronize with the central microcell when there are handoff requests from one to another. Similarly both peripheral and central microcells have to synchronize. These interactions of the different components of the network are recorded in the collaboration diagram.

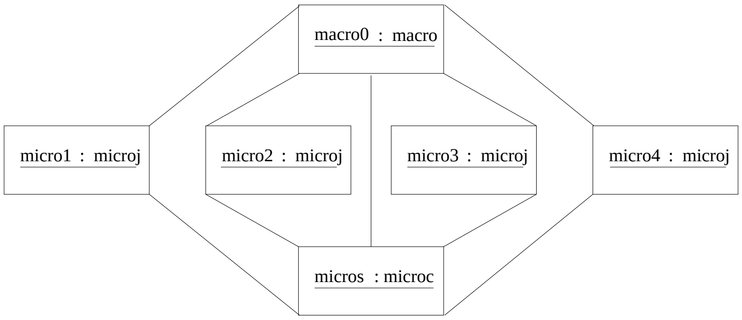

Fig. 6. The Collaboration Diagram

4.2 Processing the model

Solving this model requires using our extractor and reflector tools and the PRISM model checker. We solve the model for $c_j = 8$, $j = 0..5$. This implies that there are 48 channels shared between 6 cells. The CTMC corresponding to this model has more than a quarter of a million states (actually 262,144 states) and the longest part of the process is generating and solving this CTMC with PRISM. We used PRISM v1.3 for this with its Hybrid solution engine and its Jacobi solver. The total storage for matrix and solution vectors built for the model required 6208Kb of memory and the solver found the solution after 466 iterations. This took 45.31 seconds on a 1.6GHz Pentium IV with 256Mb of memory. The fact that this runtime is so short means that these tools can be used by a software developer with no more significant impact on development time than that spent on the edit-compile-run cycle in software development. We think that this is an encouraging indicator for this method of software performance analysis.

A screenshot showing the reflected results in the UML model can be seen in Fig. 7. On each diagram state, we now have the name of the state and, between the brackets, a performance measure related to this state. In this example we have the steady-state residence probability, expressed as a percentage, for each state.

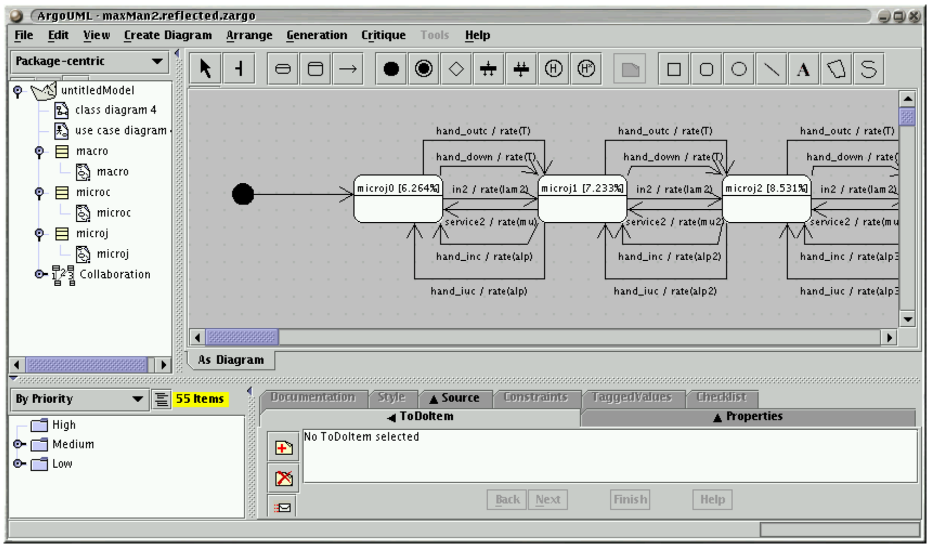

Fig. 7. The UML model with performance information added viewed in ArgoUML

4.3 Analysis and model-checking

The PRISM model checker supports the analysis of probabilistic and stochastic systems by allowing a modeller to check a logical property against a model. Several logics and several types of model are supported. The appropriate logic for continuous-time Markov chains is CSL [12] and PRISM supports this type of analysis of our models.

The syntax of CSL is:

$$\begin{aligned}\phi &::= \mathit{true} \mid \mathit{false} \mid a \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid \mathcal{P}_{\bowtie p}[\psi] \mid \mathcal{S}_{\bowtie p}[\phi] \\ \psi &::= \mathrm{X}\ \phi \mid \phi\ \mathrm{U}^{I}\ \phi \mid \phi\ \mathrm{U}\ \phi\end{aligned}$$

where a is an atomic proposition, $\bowtie \in \{<, \leq, >, \geq\}$ is a relational parameter, $p \in [0, 1]$ is a probability, and I is an interval of $\mathbb{R}$.

Paths of interest through the states of the model are characterised by the *path formulae* specified by $\mathcal{P}$. Path formulae either refer to the next state (using the X operator), or record that one proposition is always satisfied until another is achieved (the until-formulae use the U-operator).

Performance information is encoded into the CSL formulae via the time-bounded until operator (U^I) and the steady-state operator, $\mathcal{S}$.

By expressing properties of interest using path formulae we can check *interval-of-time* performability measures over our system. By expressing properties of interest with the steady-state operator we can determine *long-run* measures over the system.

5 Related work

Work which is similar in spirit to our own approach is that of Petriu and Shen [13] where a layered queueing network model is automatically extracted from an input UML model with performance annotations in the format specified by a special-purpose UML profile [14]. We do not follow the same UML profile because it is not supported by our modelling tool. Additionally, the performance evaluation technology which we deploy (process algebras and BDD-based solution) is quite different from layered queueing networks.

Another performance engineering method which is similar to ours is that of López-Grao, Merseguer and Campos [15] where UML diagrams are mapped into GSPNs which can be solved by GreatSPN. We use different UML diagram types from these authors and, again, a different performance evaluation technology. Stochastic Petri nets and stochastic process algebras have different, but complementary, modelling strengths [16].

One feature of our work which is distinctive from both of the above is the role of a *reflector* in the system to present the results of the performance evaluation back to the UML modeller in terms of their input model. We consider this to be a strength of our approach. We do not only compile a UML model into a performance model, we also present the results back to the modeller in the UML idiom.

6 Conclusions

We have described a component-based method of linking a collection of software tools to facilitate automated processing of UML performance models. The connectors in this method are the extractors and reflectors which we have developed. We have applied the tools to the analysis of a realistic model of a hierarchical cellular telephone network.

This approach to modelling allows the modeller to access a powerful and efficient solution technology without having to master the details of unfamiliar modelling languages such as process algebras and reactive modules. Our experience of using the PEPA and PRISM tools has been uniformly good.

One of the decisions which we have had to take in this work was the choice of UML diagrams and metaphors to employ. In part our choice in this was restricted by the degree of support offered by the UML modelling tool which we used (ArgoUML). However, the outcome of this was that it directed us to use familiar and well-understood parts of the UML modelling notation. One of our motivations for this work is reducing the potential for error in early stages of the performance modelling process and we consider that this outcome is supported by this influence to use the well-understood parts of UML.

We hope that we have gone some way to providing automated support for computing simple performance measures and to circumventing an unnecessary notational hurdle if this was acting as an impediment to the understanding and uptake of modern performance analysis technology.

The dependability and safety of computer-based systems is a complex issue with many opposing and sometimes conflicting aspects. In this paper we have focused on quantitative aspects of system dependability taking the view that it is sometimes the case that quantitative analysis takes second place to qualitative analysis of systems. Well-engineered, safe systems need to deliver reliable services in a timely fashion with good availability. For this reason, we view quantitative analysis techniques as being as important as qualitative ones.

We have recently developed an extension of the PEPA stochastic process algebra where PEPA components are used as coloured tokens in a stochastic Petri net. The resulting formalism is called PEPA nets [17]. Our future work is to integrate the PEPA nets formalism with our extractor and reflector tools. Given an extended UML tool which supports the forthcoming UML 2.0 design we would be able to map the extended UML 2.0 activity diagrams onto PEPA nets for analysis purposes. The activity diagrams in UML 2.0 are given a semantics which is based on Petri nets and queueing theory and are intended for analyses such as ours. An algorithm translating PEPA nets models into the PEPA formalism has already been developed and implemented [18]. Using this it would be possible to take extended activity diagrams through to analysis by PRISM using the method followed in this paper.

Acknowledgements: The authors are supported by the DEGAS (Design Environments for Global ApplicationS) project IST-2001-32072 funded by the FET Proactive Initiative on Global Computing. The authors thank Gethin Norman and David Parker of the University of Birmingham for the implementation of the PEPA process algebra combinators in the PRISM model checker. Jane Hillston and David Parker provided helpful comments on an earlier draft of this paper.

References

1. Tigris.org project. ArgoUML: A modelling tool for design using UML. Web page and documentation at `http://argouml.tigris.org/`, 2002.
2. 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.
3. M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In T. Field, P. Harrison, J. Bradley, and U. Harder, editors, *Proc. 12th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation (TOOLS'02)*, volume 2324 of *LNCS*, pages 200–204. Springer, 2002. `http://www.cs.bham.ac.uk/~dxp/prism/`.
4. Object Management Group. Unified Modeling Language, v1.4, March 2001. OMG document number: formal/2001-09-67.
5. J. Hillston. *A Compositional Approach to Performance Modelling.* Cambridge University Press, 1996.
6. R. Alur and T.A. Henzinger. Reactive modules. *Formal Methods in System Design: An International Journal*, 15(1):7–48, July 1999.

7. D. Parker. *PRISM 1.3 User's Guide*. University of Birmingham, February 2003. `http://www.cs.bham.ac.uk/~dxp/prism`.
8. F. Somenzi. *CUDD: CU Decision Diagram Package*. Department of Electrical and Computer Engineering, University of Colorado at Boulder, February 2001.
9. Robin Milner, Mads Tofte, Robert Harper, and David MacQueen. *The Definition of Standard ML: Revised 1997*. The MIT Press, 1997.
10. M.D. Kulavaratharasah and A.H. Aghvami. Teletraffic performance evaluation of microcellular personal communication networks (PCN's) with prioritized handoff procedures. *IEEE Transactions on Vehicular Technology*, 48(1):137–152, January 1999.
11. I. Katzela and M. Naghshineh. Channel assignment schemes for cellular mobile telecommunication systems: A comprehensive survey. *Proceedings of the IEEE*, 82(9):1398–1430, 1994.
12. A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Verifying continuous time Markov chains. In *Computer-Aided Verification*, volume 1102 of *LNCS*, pages 169–276. Springer-Verlag, 1996.
13. D.C. Petriu and H. Shen. Applying the UML performance profile: Graph grammar-based derivation of LQN models from UML specifications. In A.J. Field and P.G. Harrison, editors, *Proceedings of the 12th International Conference on Modelling Tools and Techniques for Computer and Communication System Performance Evaluation*, number 2324 in Lecture Notes in Computer Science, pages 159–177, London, UK, April 2002. Springer-Verlag.
14. B. Selic, A. Moore, M. Woodside, B. Watson, M. Bjorkander, M. Gerhardt, and D. Petriu. Response to the OMG RFP for Schedulability, Performance, and Time, revised, June 2001. OMG document number: ad/2001-06-14.
15. J.P. López-Grao, J. Merseguer, and J. Campos. From UML activity diagrams to stochastic Petri nets: Application to software performance analysis. In *Proceedings of the Seventeenth International Symposium on Computer and Information Sciences*, pages 405–409, Orlando, Florida, October 2002. CRC Press.
16. S. Donatelli, J. Hillston, and M. Ribaudo. A comparison of Performance Evaluation Process Algebra and Generalized Stochastic Petri Nets. In *Proc. 6th International Workshop on Petri Nets and Performance Models*, Durham, North Carolina, 1995.
17. S. Gilmore, J. Hillston, L. Kloul, and M. Ribaudo. PEPA nets: A structured performance modelling formalism. *Performance Evaluation*, 2003. Special issue of selected papers from the Proceedings of the 12th International Conference on Modelling Tools and Techniques for Computer and Communication System Performance Evaluation. To appear.
18. S. Gilmore, J. Hillston, L. Kloul, and M. Ribaudo. Performance modelling with PEPA nets and PRISM. In *Proceedings of the Second PASTA workshop*, pages 23–39, Edinburgh, Scotland, June 2003.