A Software Performance Evaluation Approach using Stochastic Process Algebra Tools

Osman Salem
IRIT
Université Paul Sabatier
Toulouse, France

Abdelmalek Benzekri
IRIT
Université Paul Sabatier
Toulouse, France

Abstract

In this paper we present an integrated approach to the functional and performance analysis of software modeling provided by Stochastic Process Algebra (SPA). Three SPA languages have been appeared in the last decade and are widely used for software architecture and performance evaluation. We present these languages with the difference between them and we show by an example the methodology which must be used for an integrated analysis.

Keywords: Stochastic process algebra; Performance evaluation; Markov chain; DiffServ.

1. Introduction

Computer software can be described using classical process algebra which provide a means for constructing an abstract model of the software in question [1]. This model is used only to establish the correct functional behaviour of complex software by deriving qualitative properties such as freedom from deadlock or livelock[1] [2]. In the past years, many languages and tools have been appeared for modelling design and they are used in software architecture instead of the box and line diagram, despite that these tools deal only with the functional aspects of software architectures. So, the software performance evaluation was in a separate phase, after the fully design and implementation of the model. Consequently, if the performance is detected to be poor, the software will be redesigned with negative consequences for both design costs and time lost, where the need of integrating the software performance analysis into the design process.

Stochastic Process Algebras (SPA) have been developed for this purpose. It is a formal specification technique which extends classical process algebras via the inclusion of timing information by using random variables in the generated models, in order to express the durations of an activity [2,3]. Once the model has been defined and parameterised, it can be used to investigate numerically the performance parameters.

The formal description of the software architecture makes available a precise document describing the structure of the system to all the people involved in the design, implementation and maintenance. So, modelling in SPA offers flexibility, safety, economy and extended facilities to system design.

This paper is organised as follows. In the second section we recall the syntax and the semantics of existing Markovian SPA languages with a comparison between them. In the third section we describe the methodology used by SPA for integrating performance and functional analysis. Finally, conclusions work is presented.

2. Stochastic Process Algebra

Stochastic Process Algebra are formal descriptions techniques used to describe the functionality of concurrent and distributed systems and to analyze their related performance [2, 3, 4]. Stochastic analysis of systems is generally performed using either queueing models or Petri Nets. Several SPA languages have been appeared in the literature, these include PEPA [5, 6], TIPP [7] and EMPA [8]. These languages have been introduced as an extension to the calssical process algebra like CCS [9] and CSP [10]. They are abstract languages constructed from a small set of powerful operators where it is possible to construct algebraic models whose key features are: compositionality (which allow the designer to built a complex model from smaller ones by means of languages operators, and to study the behaviour of each component

[1] State in LTS from which there is no possible exit transition represent deadlocks and a subset of states in the graph from which no transition is possible indicates a livelock.

separately), and abstraction (which allows the internal details of a system description to be hidden from an external observer at analysis time). In these languages, softwares and systems are modeled as collections of entities, called agents or processes, which execute actions. These actions are the building blocks of these languages and they are used to describe sequential behaviours which may run concurrently by synchronisations or by communications between them.

These languages propose the same approach to performance modelling: a random variable is associated with each action, representing its duration. This random variable is assumed to be exponentially distributed and this leads to a clear relationship between the process algebra model and a Continuous Time Markov Chains (CTMC). Via this underlying CTMC derived from the model semantic description [2], different types of analysis may be performed, like steady-state and transient probability distribution. This analysis is done through the compilation of the infinitesimal generator matrix of the Markov process like we will explain in section 3.

Work on SPA has increased rapidly over the last few years. Here, we describe the syntax and semantic for PEPA, TIPP and EMPA:

2.1 PEPA

Performance Evaluation Process Algebra (PEPA), has been developed at the university of Edinburgh by Jane Hillston [6]. PEPA is Markovian process algebra by means that associate an exponential delay to each activity in a process description. This language is supported by a tool called PEPA Workbench. A PEPA component is represented syntactically according to the following syntax:

$$P = (a,r).P \mid P + Q \mid P < L > Q \mid P/L \mid A$$

Prefix (.): the component (a,r).P performs the activity "a" after exponential distributed delay with rate "r" denoted by its distribution function $F(t)=1-e^{-rt}$, then behaves like P. The prefix combinator specifies sequential behaviour of activities. In some cases, the rate of an action is outside the control of this component, such actions are carried out jointly with another component, so this component playing a passive role. This is recorded by the distinguished symbol T (called top).

Choice (+): a choice between two possible behaviours is represented as the sum of the possibilities such P + Q behaves either like P or like Q. A race condition is assumed to govern the behaviour of simultaneously enabled actions, the action with the least duration is executed.

Cooperation <L>: the component P <L> Q specifies interaction between components on a given set of action types. This means that for action types outside the set L, P and Q proceed independently, but for action types in L, P and Q must cooperate and synchronise over given gates listed in L, so actions in L require the simultaneous involvement of both components. Shared activities should proceed at the rate of the slower of the two participating components. If the set of action "L" is empty, then there is no interaction between P and Q, and in this case, cooperation reduces to parallel composition, written <> or || and these components will behave in an interleaved manner. It is clear that if a component is behaviourally independent, then it may be studied in isolation without affecting its behaviour.

Hiding (/): this operator hides the behaviour of components from an external observer and transforms it to an internal action (i in LOTOS [13]). So the component P/L behaves like P, excepting that any activity with type in L is not visible to an external observer.

Constants (A): constants allow names to be assigned to components and give the variable on the left hand side of the assignment operator the behaviour of the component on the right hand side.

Distinct precedence were assigned to the PEPA: the hiding operator was given highest precedence with prefix next, followed by cooperation. The choice operator was given the lowest precedence.

2.2 TIPP

TImed Processes and Performability evaluation (TIPP), was developed at the university of Erlangen by the group of Prof Ulrich Herzog [7,11]. The semantics rules of this language can be described by the following expression:

$$P = Stop \mid (a,r).P \mid P + Q \mid P \|_s Q \mid P\backslash a \mid rec\ X : P \mid X$$

The term Stop denotes the terminated process. Prefix operator ".", Choice "+", parallel (cooperation) "$\|_s$" and hiding "\" operators have the same functions like in PEPA. Passive actions are represented with rate 1 instead of T which is used in PEPA. The recursion operator "rec" is used to express the recursion equation when X reappears in P expression and finally the constant component which allow names to be assigned for components.

TIPP is supported by TIPPtool [12] which is closely related to LOTOS [13]. It allows one to model immediate activities in addition to exponentially timed actions and its semantics rules can be expressed by the following:

$$P = Stop \mid (a,r);P \mid (tau,r);P \mid a;P \mid tau;P \mid exit \mid P >> Q$$
$$P [> Q \mid P [\ ] P \mid P \,|||\, P \mid P \mid [a_1,a_2, \ldots a_n] \mid Q$$
$$P[a_1,a_2, \ldots a_n](r_1,r_2, \ldots r_n)$$
$$hide\ a_1,a_2, \ldots a_n\ in\ P$$

"tau" is used to express an unobservable action. The prefixed process a;P behaves like P after "a" is immediately executed. Exit is used to express successful termination. The enable operator P >> Q is used to allow the execution of process Q if process P is successfully terminated. The disabled operator P [> Q allows process Q to interrupt the execution of process P. The choice operator is represented by "[ ]" instead of "+". The pure interleaving operator, which is used when the synchronisation list between two processes is empty is represented by "|||" and by $|[a_1,a_2, \ldots a_n]|$ if the synchronisation list contains $a_1,a_2, \ldots a_n$.

2.3 EMPA

Extended Markovian process algebra (EMPA), has been developed at the university of bologna by Marco Bernardo [8]. Its development has been strongly influenced by the stochastic process algebra PEPA and TIPP. Those are witnessed by the fact that in EMPA there are three different kinds of actions: exponentially timed actions, passive actions and prioritised weighted immediate actions. The syntax of EMPA can be summarized by the following expression:

$$P = 0 \mid <a,\lambda>.P \mid <a,*>.P \mid <a,\infty_{L,W}>.P \mid P/L \mid P\backslash L \mid P[\varphi] \mid P + P \mid P \|_s P \mid A$$

Since the null term "0" is the deadlock operator, the prefix operator is "<a,λ>.P", the functional abstraction operator or hiding operator is "P/L", the functional relabelling operator is "P[φ] ", the choice operator "+", the parallel composition operator "$\|_s$" and the constant are the same operators that we have seen in the previous SPA languages. This SPA language is supported by a software tool called TwoTowers.

A passive action whose duration is unspecified, is represented in EMPA by "*" in order to model activities waiting for synchronisation. An immediate action is represented by "<a ,$\infty_{L,W}$>._" where L is used to express the priority level and W is used for the probability weight. In the race condition, immediate actions take precedence over exponentially distributed actions and over other immediate actions having small priority level with respect to their level. If two immediate actions have the same priority level, they will be executed according to the probability associated with each one. The temporal restriction operator "P\L" prevent the execution of passive actions who are listed in L.

2.4 Comparison

The main difference between these stochastic process algebras is related to the synchronisation of stochastic actions of two concurrent component of the software model [3]. In order to illustrate this difference, we take the following behaviour expression:

$$(a,\lambda).B \parallel a \parallel (a,\mu).C = (a, \lambda*\mu).(B \parallel a \parallel C)$$

And we will show the solution adopted by these languages for finding the function "*" (law of cooperation) that makes the equality in the previous equation hold.

In PEPA, the function "*" correspond to the minimum of the related rates, in order to express that synchronisation between these components is determined by the rate of the slowest one. In contrast TIPP adopts another solution, where "*" is interpreted as the ordinary multiplication, but the operational intuition behind the choice of multiplication for this function is not at all obvious and no useful stochastic interpretation of this solution other than algebraic simplicity. EMPA deals with synchronisation by adopting the client/server (or master/slave) model, where the server determines the rate of service and the client plays a passive role with this respect. This synchronisation discipline imposes that action (a,λ) can be synchronised with action (a,μ), if and only if min(λ,μ) is unspecified and the rate of the resulting action is given by max(λ,μ), or in other words, in synchronisation at most one action must be active and all the other involved actions must be passive.

3. EMPA specification for a DiffServ router

In this section we illustrate an example that should highlight the expressive power of SPA languages in performance evaluation. We have chosen a simple DiffServ router which deals with 2 classes of services, one for handling packets with high priority level and the other for packets with low priority level. This router can be modelled by a server with a limited buffer, and packets by clients which wait for service in the buffer if the server is busy. This queueing model is represented by M/M/1/N in kendall notation if we suppose that both arrival and departure rates are Markovian. H and L will be used for indexing actions and rates for high and low priority levels respectively.

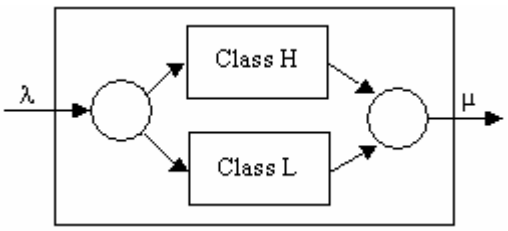

Figure 1. DiffServ router model

EMPA language provide an algebraic method for performance evaluation with the model description instead of manual method (full scan of the CTMC diagram) used by other tools. This is the reason why this

language is chosen through this queueing modelling example.

We assume that the arrival rate for the two different classes are λ_H, λ_L respectively, and the service rate are equal (only for simplicity of equation due to lack of space). The priority mechanism affects the service discipline, i.e that possible preemption of low priority packet being served can be exercised in addtion to dropping low priority packets when the queue become full. This queueing system (QS) can be modelled as follows:

$DiffServ = Arrival \|_A (Queue_{0,0} \|_D Pre\_emp\_Server);$
$A = \{arr_H, arr_L\};$
$D = \{deliver_H, deliver_L\};$
$Arrival = <arr_H,\lambda_H>.Arrival + <arr_L,\lambda_L>.Arrival;$
$Queue_{0,0} = <arr_H,*>.Queue_{1,0} + <arr_L,*>.Queue_{0,1};$
$Queue_{i,0} = <arr_H,*>.Queue_{i+1,0} + <arr_L,*>.Queue_{i,1} + <deliver_H,*>.Queue_{i-1,0};$ (if $0<i<N-1$)
$Queue_{0,j} = <arr_H,*>.Queue_{1,j} + <arr_L,*>.Queue_{0,j+1} + <deliver_L,*>.Queue_{0,j-1};$ (if $0<j<N-1$)
$Queue_{i,j} = <arr_H,*>.Queue_{i+1,j} + <arr_L,*>.Queue_{i,j+1} + <deliver_H,*>.Queue_{i-1,j} + <deliver_L,*>.Queue_{i,j-1};$ (if $i,j>0$ and $i+j<N-1$)
$Queue_{N-1,0} = <deliver_H,*>.Queue_{N-2,0};$
$Queue_{0,N-1} = <deliver_L,*>.Queue_{0,N-2} + <arr_H,*>.<loose_L,\infty_{2,1}>.Queue_{1,N-2};$
$Queue_{i,j} = <deliver_H,*>.Queue_{i-1,j} + <deliver_L,*>.Queue_{i,j-1} + <arr_H,*>.<loose_L,\infty_{2,1}>.Queue_{i+1,j-1};$ (if $i,j>0$ and $i+j = N-1$)
$Pre\_emp\_Server = <deliver_H,\infty_{2,1}>.ServePLHigh + <deliver_L,\infty_{1,1}>.ServePLLow;$
$ServePLHigh = <serve,\mu>.Pre\_emp\_Server;$
$ServePLLow = <serve,\mu>.Pre\_emp\_Server + <deliver_H,\infty_{2,1}>.<serve,\mu>.ServePLLow;$

When the model is loaded in TwoTowers [14], its descriptions are syntactically and semantically analyzed using a parser for detecting errors, then TwoTowers will find all possible states and transitions or in another word the labelled transition diagram or derivation graph.

The Labelled Transition System (LTS) which is called in EMPA integrated interleaving semantics model, is a graph (described by a list) where each state is represented by a node and transitions between states are represented by arrows labelled with the actions that fire this transition and the rates of the corresponding actions as a list (figure 2.A). From such an integrated model, two other models can be derived: the functional model which can be obtained by discarding the performance related information e.g the rate of actions (see figure 2.B) and can be used to detect the functional errors such as a deadlock or a livelock, and the performance model or the Markovian (figure 2.C) semantic model is obtained from integrated model by discarding action type and hiding (discarding) the immediate transition and the related source states (vanishing states). The reason of removing immediate transitions and their related source states is that the sojourn time in those states is zero, so they are irrelevant from the performance view point.

The objective of this modelling study is to calculate the steady-state and transient probability distribution for the system which will be used to derive the performance measures such as system throughput and resource utilization.

The performance semantic model (figure 2.C) can be used for constructing the infinitesimal generator matrix defined by these following expressions:

$$q_{ij} = \lambda_{ij} \qquad \text{for } i \neq j$$

$$q_{ii} = -\sum_{i=1, i\neq j}^{N} q_{ij} \quad \Rightarrow \quad \sum_{i=1}^{N} q_{ij} = 0$$

The transient state can be found by solving the equation: $(d\pi/dt) = \pi(t).Q$, and at the equilibrium, the probability distribution vector will be unchangeable $(d\pi/dt) = 0$, so using the steady-state probability we obtain $\pi.Q = 0$, where π is the distribution probability vector of the model at the steady state [2], so $\sum_{i=1}^{N} \pi_i = 1$.

But in using TwoTowers, no need to waste our time in calculating Q matrix because TwoTowers will give us the steady state and the transient state probability. So given

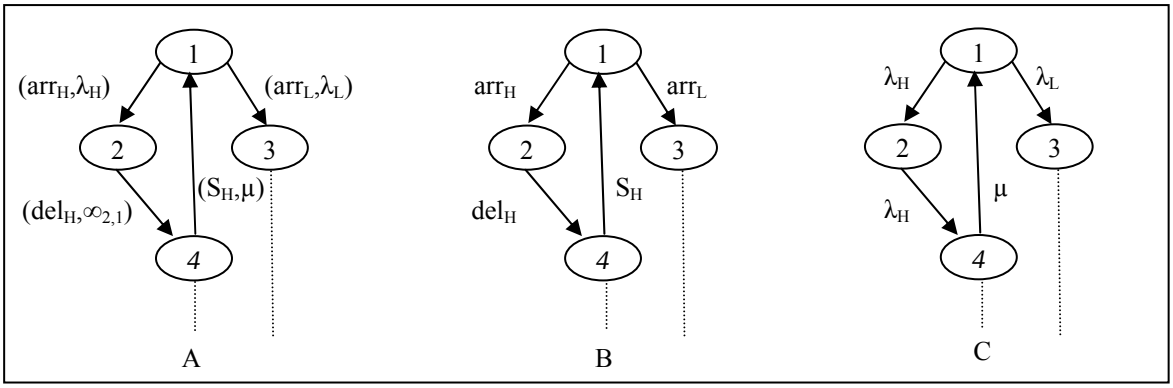

Figure 2. Label Transitions System

the CTMC diagram (performance model) and the value of probability distribution in steady and transient state, we can evaluate the performance of the system by using queueing system theory. For example, the throughput is defined as the mean number of customers served per time unit, and is given by the service rate multiplied by the stationary probability of being in a state where service can be provided. It is given by the following formula:

$$T = \sum_{i=2}^{N} \mu.\pi_i = \mu.(1-\pi_1)$$

Where μ and π_1 are given, so the throughput can be computed by a simple multiplication.

The utilization rate of the QS is defined by the fraction of time during which the server is busy, and which is the sum of the stationnary probabilities of states where there is at least one packet. It is given by the following formula:

$$U= \sum_{i=2}^{N}(1.\pi_i) = \sum_{i=2}^{N} \pi_i = 1-\pi_1$$

The vector distribution probability π_i for all states is given by TwoTowers and the utilisation rate can be calculated by a simple addition.

The mean number of customer which is given by:

$$E[N] = \sum_{i=2}^{N}(i.\pi_i)$$

It can be computed using the CTMC diagram in order to determine the number of customer in each state, but a full scan to the performance semantic model graph will be excendingly expensive for finding the number of paquets in each state (especialy if we have a big number of states) and this technique must be avoided. This is why EMPA take into account the performance aspects of a system in the early stages of its design (with algebra description) where designers can assign reward to each action. Actions with reward will be specified according to the following syntax:

$$A= <a,r,y,b>.A$$

Where y is the yield reward and b is the bonus reward. A performance measure for a CTMC can be specified by attaching a yield reward y_i to every state i, which expresses the rate at which reward is accumulated at state i, and by attaching a bonus reward $b_{i,j}$ to every transition from state i to state j, which expresses the instantaneous gain due to the execution of the transition from state i to state j. Readers interested about yield and bonus reward can refer to [14].

Given yield and bonus rewards, the corresponding stationary performance measure can be computed in EMPA according to the following equation:

$$\sum_{i=1}^{N} y_i .\pi_i + \sum_{i=1}^{N}\sum_{j=1}^{N} b_{ij}.\pi_i .q_{ij}$$

and for transient phase, we use $\pi_i(t)$ in place of π_i in this equation. Many of the performance measures can be obtained using this formula, for exemple: if we want to compute the throughput, we must replace every action of the form <serve,μ> with <serve,μ,μ,0> (e.g $y_i = \mu$ and $b_{ij} = 0$) for obtaining the following equation:

$$\sum_{i=1}^{N} y_i .\pi_i + \sum_{i=1}^{N}\sum_{j=1}^{N} b_{ij}.\pi_i .q_{ij} = \sum_{i=2}^{N} \mu.\pi_i$$

EMPA will take into account all states that provides the service s and will assign a reward to them. Like we know from CTMC of this router, that the first state is the only state that can not execute action serve, this is why the index in the answer begin from 2 and the first state will not be taken into account by the tool.

If we want to compute the utilization of the QS, we must replace every action of the form <serve,μ> with <serve,μ,1,0> (e.g y_i=1 and b_{ij}=0) in order to obtain the following equation:

$$\sum_{i=1}^{N} y_i .\pi_i + \sum_{i=1}^{N}\sum_{j=1}^{N} b_{ij}.\pi_i .q_{ij} = \sum_{i=2}^{N} 1.\pi_i$$

Performance results for our model specification are computed using queueing theory and EMPA reward technique. Like expected and like we have shown in the previous equations, these results are equal.

In table 1 we summarize the results obtained from EMPA supported tool (TwoTowers) by varying the capacity of the previous queueing system from 3 to 6, where $\lambda_H = \lambda_L = 2$ and $\mu = 4$. The columns show respectively, the capacity of the system, the number of states of the integrated interleaving semantic model, the number of tangible states (which have only outgoing exponentially transitions and which appear in the Markov chain), the number of vanishing states (which have immediate transitions), the number of deadlock states, the number of transitions, the number of exponentially timed transitions, the throughput of the system in stationary phase and in transient phase (at time = 3 second), the utilization rate at stationary phase and at transient phase (at time = 3 second).

N	states	ts	vs	ds	transitions	ett	it	throughput	throughput at t = 3s	utilisation	utilisation at t = 3s
3	29	16	13	0	51	38	13	3.10	3.09	0.78	0.77
4	47	25	22	0	85	63	22	3.28	3.23	0.82	0.80
5	69	36	33	0	127	94	33	3.39	3.30	0.85	0.82
6	95	49	46	0	177	131	46	3.47	3.33	0.86	0.83

Table 1. Results of the analysis of the DiffServ router with 2 classes

4. Conclusions

In this paper we have shown the methodology used by SPA languages for modeling and analyzing both functional and performance characteristics of a system. This is done in two phases:

- The first phase requires the designer to specify the system in the stochastically timed process algebra, in order to perform a behavioral analysis such as deadlock free.
- The second phase consists in deriving from the algebraic representation of a system the performance characteristics.

We have applied TwoTowers (which is a tool supporting EMPA SPA language) for modelling and analyzing a DiffServ router with two classes and we illustrate how this language integrates functional verification and performance evaluation.

5. References

[1] Marco Bernardo, On the Formalization of Architectural Types with Process Algebras, Proc. ACM/IEEE Int. Conf. on Fundamentals of Software Engineering, 2000.

[2] Abdelmalek Benzekri, Qualitative and Quantitative Evaluation using Process Algebra, The 17th International Symposium on Computer and Information Sciences, Orlando, Florida USA, pp. 415-418, 28 October 2002.

[3] Ed Brinksma and Holger Hermanns, Process Algebra and Markov Chains, Lectures on Formal Methods and Performance Analysis, pp. 183 – 231, Nijmegen, 2001.

[4] Jane Hillsotn and Marina Ribaudo, Stochastic Process Algebra: A New Approach To Performance Modeling, Modeling and Simulation of Advanced Computer Systems, Gordon Breach, 1998.

[5] Jane Hillston, A Compositional Approach to Performance Modelling, Cambridge University Press, 1996.

[6] Jane Hillston, PEPA Performance Evaluation Process Algebra, Technical Report of Computer Science, Edinburgh University, March 1993.

[7] H. Hermanns, V. Mertsiotakis, A Construction and Analysis Tool Based on the Stochastic Process Algebra TIPP, 2nd Int. Workshop on Tools and Algorithms for the Construction and Analysis of Systems, 1996.

[8] M.Bernardo, A Tutorial on EMPA: A Theory of Concurrent Processes with Nondeterminism, Priorities, Probabilities and Time, in Theoretical Computer Science, pp. 1-54, July 1998.

[9] Robin Milner, Communication and Concurrency, Prentice-Hall, 1989.

[10] C.A.RHOARE, Communicating Sequential Processes, Prentice-Hall, 1985.

[11] U. Herzog, Performance Evaluation and Formal Description, IEEE CompEuro 91, IEEE Computer Society Press, May 1991.

[12] U. Klehmet, V. Mertsiotakis, TIPPtool: Timed Processes and Performability Evaluation, User's Guide, Technical Report IMMD VII-3/98 University of Erlangen-Nurnberg, January 1998.

[13] T. Bolognesi, E. Brinksma, Introduction To The ISO Specification Language LOTOS, Computer Networks and ISDN Systems 14, Elsevier Science Publishers B.V.(North Holland), pp. 25-59, 1987.

[14] M. Bernardo, An Algebra-Based Method to Associate Rewards with EMPA Terms, the 24th Int. Coll. On Automata Languages and programming (ICALP 97), 1997.