Proceedings of *XXXVII Brazilian Conference of Operational Research (SBPO), Gramado, RS, Brazil*, 27th to 30th September 2005

ON SPECIFYING COMPLEX SYSTEMS THROUGH STOCHASTIC STATECHARTS FOR PERFORMANCE ANALYSIS

Carlos Renato Lisboa Francês[1]
Nandamudi Lankalapalli Vijaykumar[2]
Marcos José Santana[3]
Solon Venâncio de Carvalho[2]
Regina Helena Carlucci Santana[3]

[1]Departamento de Engenharia Elétrica
Universidade Federal do Pará
Belém, PA
rfrances@ufpa.br

[2]Laboratório Associado de Computação e Matemática Aplicada – LAC
Instituto Nacional de Pesquisas Espaciais - INPE
São José dos Campos, SP
{solon, vijay}@lac.inpe.br

[3]Instituto de Ciências de Matemática e de Computação – ICMC
Universidade de São Paulo – USP
São Carlos, SP
{mjs, rcs}@icmc.sc.usp.br

Abstract

Performance evaluation of complex systems is an essential issue nowadays. Evaluation may be obtained from both simulation and analytical approaches. However, one of the present important topics is towards the specification of complex systems which are reactive by nature. Specification techniques include state-transition diagrams, queueing networks, Petri nets and recently Statecharts have been introduced as another approach to represent complex systems and obtain performance analysis based on analytical methods. The paper discussed here is also based on Statecharts representation by providing stochastic features to the specification technique. The idea is to associate the specification technique to a solution that generates performance measures. Simulation solution is commented in a case study of a distributed programming environment based on PVM (Parallel Virtual Machine).

Key words: Performance models, Statecharts specification, Analytical Approach, Markov chains, Simulation

1. Introduction

In many a situation while evaluating a system, decision-makers have to predict the system saturation so that it is possible to determine the best cost effective means to avoid it or at least delay this saturation as much as possible. One must have a notion of how the system behaves both in best and worst cases. This is generally provided by the expertise of the system administrator. Another alternative is to use performance evaluation techniques [Jain, 1991]. These techniques can be classified into two main categories, which are measuring and modeling. Usually, measurements, benchmarking and prototyping fall under the category of measuring. These techniques are very useful to be applied on systems that have already been built and they can provide accurate information for analyzing their performance.

The second category mentioned above, modeling, can also be used for providing performance analysis. Modeling is a complex process that starts usually with a high-level specification (either graphical or non-graphical) and ends up with the presentation of performance measurements. In order to obtain these measurements, two approaches can be utilized: analytical and simulation solutions. Analytical approach usually associates the system specification to a mathematical model such as Markov chains or queuing theory. Simulation approach, on the other hand, is a computer program that reproduces the system behavior. Figure 1 shows the modeling process and the phases involved to conduct a performance evaluation.

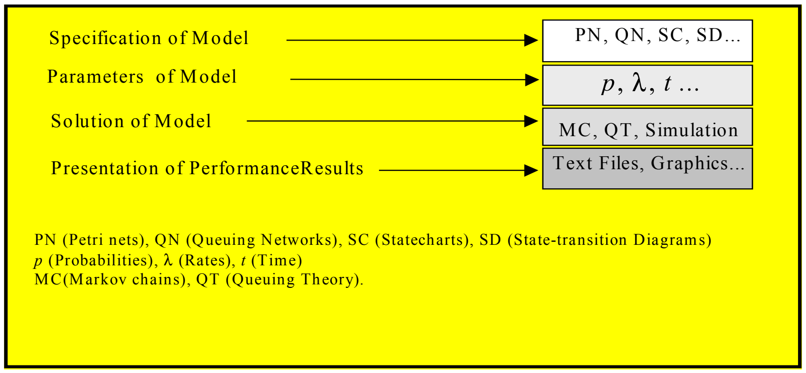

Specification of Model
PN, QN, SC, SD...
Parameters of Model
p, λ, t ...
Solution of Model
MC, QT, Simulation
Presentation of PerformanceResults
Text Files, Graphics...
PN (Petri nets), QN (Queuing Networks), SC (Statecharts), SD (State-transition Diagrams)
p (Probabilities), λ (Rates), t (Time)
MC(Markov chains), QT (Queuing Theory).

Figure 1. Phases of a Modeling Process.

Present day models, whose performance is to be evaluated, are complex as they involve parallelism, synchronization and interdependence of subsystems. They have been called as ***reactive systems*** as they are based on states and events. Dynamics of such systems are observed by change of one state into another based on responses to stimuli received by external as well as internal means. However, the main concern is how to describe the behavior of reactive systems in a clear and realistic way and at the same time maintaining a rigorous formal basis that can be computationally handled [Harel, 1987]. In order to model these systems, ideas of parallelism, resource sharing, synchronization, interdependence, hierarchy and randomness (random perturbations) are to be considered.

Some graphical techniques (for the purpose of this paper) that are generally associated to stochastic processes (due to their precise mathematical definition added to the fact that they can be computationally handled) may be mentioned: Queuing networks [Kleinrock, 1976], Petri nets [Peterson, 1981], and Statecharts [Harel, 1987], [Vijaykumar, 1999]. The idea of associating Statecharts to Markov chains in order to obtain performance measurements is discussed in [Vijaykumar, 1999] and [Vijaykumar et al, 2002].

Usually systems whose performance has to be evaluated have some peculiarities such as parallelism, communication and hierarchy within the components. Stochastic Statecharts use these features that already exist in Statecharts and associate the specification to a solution method in order to obtain performance measures.

In this paper stochastic features are embedded into the original Statecharts in order to make the specification for performance analysis more convenient. An example of a case study involving process scheduling in a computer network is explored to illustrate the use of Stochastic Statecharts.

The paper is organized as follows: Section 2 discusses very briefly some main features of Statecharts and how the events are considered for the purpose of generating steady-state probabilities through a Markov chain. Section 3 is dedicated to show the syntax and semantics of Stochastic Statecharts. Section 4 presents a case study on process scheduling in a distributed environment. Finally the paper ends with conclusions in Section 5.

2. Statecharts Specification

Statecharts are graphical-oriented and are capable of specifying reactive systems. They have been originally developed to represent and simulate real time systems [Harel, 1987]. Moreover

Statecharts come with a strong formalism [Harel et al., 1987] and [Harel & Politi, 1998] and their visual appeal along with the potential features enable considering complex logic to represent the behavior of reactive systems. They are an extension of state-transition diagrams and these diagrams are very much improved with notions of hierarchy (depth), orthogonality (representation of parallel activities) and interdependence (broadcast-communication).

States are clustered by means of representing depth. With this feature it is possible to combine a set of states with common transitions into a macro-state also known as super-state. Super-states are usually organized before being refined into further sub-states thus enabling a top down approach. State refinement can be achieved by means of *XOR* decomposition and *AND* decomposition. The former decomposition may be used whenever a encapsulation is required. When a super-state in a high level of abstraction is active, one (and only one) of its sub-states is indeed active. The latter approach is used to represent concurrency. In this case when a super-state is active, all of its sub-states are active. One more type of state can be mentioned that is *BASIC* which means that there no refinements from this type of state.

In Statecharts the global state of a given model is referred to as a *configuration* that is the active basic states of each orthogonal component. Details of definition of each element as well as the main features are described in [Harel, 1987], [Harel et al., 1987] , [Harel and Namaad, 1996] and [Harel and Politi, 1998].

However, a brief discussion of events considered for performance evaluation [Vijaykumar, 1999] and [Vijaykumar et al, 2002] is in order. Events have been classified into two categories: internal events and external events. Internal events are those that take zero time when they are enabled. They are also known as immediate events as the reaction to these take place immediately. Statecharts have such in-built events: *true(condition), false(condition), entered(State), exit(State)*. The basic element *action* as an event that can influence some other orthogonal component is also considered as an immediate event. This means that whenever an event is associated as action, a reaction to this event is immediate. External events are stochastic events (where time between their activation and their occurrences follow a stochastic distribution) that have to be externally stimulated to yield new configurations. In order to make the association of a Statecharts model with a Markov chain (which consists of converting the Statecharts specification into a Markov chain), the only type of events considered are stochastic events. This means that the time between activation and occurrence of events follows a stochastic distribution. In particular, for Continuous-Time Markov Chains, this distribution has to be exponential.

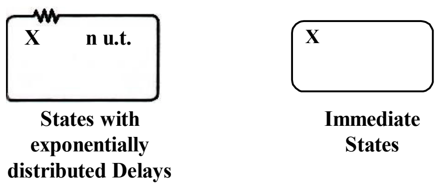
3. Stochastic Statecharts – Syntax, Semantics and Templates

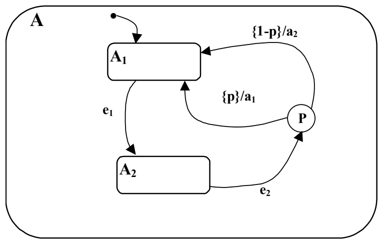
In this section, a stochastic formalism to Statecharts is introduced by some points of the original semantics and syntax proposed in [Harel et al., 1987]. This introduction enables the use of Statecharts features in performance evaluation [Francês et al, In Press]. More specifically templates are proposed to be used in specifying queuing systems associating with a possible solution.

3.1. Syntax Modifications

Two main modifications have been added to the original specification: states with exponentially distributed delays and inclusion of probabilities in a transition. The first modification considers delays as random variables (*interarrival time* and *service time*). Thus a state with delay can be a source that, for example, generates clients yielding an exponential distribution or a certain server that grants a service to customers according to an exponential distribution. Figure 2 (a) shows the graphical notations for two categories of states: immediate states where an activity within it takes zero time, i.e., a change to another state occurs immediately; states with delay where an activity takes an exponential time.

The second modification refers to the necessity of associating a probabilistic value for each possible path followed by a client. This situation forces the customer to opt for a path among various possibilities. The selection by probability is represented by means of a connector (represented as a circle and already defined in [Harel, 1987]) with the letter *P* inside it. Figure 5 (b) illustrates this aspect.





(a) (b)

Figure 2. Syntax Adaptations: (a)State with delay and immediate state; (b)Selection for Probability.

Formally, the probabilistic condition is defined as:

- $\exists$ P={p_1, p_2, ..., p_n}, where P is the set of probabilities associated to the events and p_i is a point within the probability space defined by P $\therefore$ $p_i \in$ P, with i=1,2,...,n;
- $0 \leq p_i \leq 1, \forall\, i = 1, 2, \ldots, n$;
- $\sum_{i=1}^{n} pi = 1$;

Thus, it is assumed that the probability connector $\subset$ condition connector C $\therefore$ $p_i \in$ C (C is the set of conditions defined by Harel [Harel et al., 1987]).

By extension, if $c \in$ C and $p_i \in$ C, then *ev(c)* and *ev(p_i)* are *events associated to a condition c* and *event conditioned to a probability p_i, respectively*. In order to distinguish the difference between a probabilistic condition and a regular condition, the notation *ev{pi}* is used in place of *ev(c)*, according to the one proposed in [Vijaykumar, 1999] and implemented in [Vijaykumar et al, In Press].

3.2. Statecharts Semantics: Steps, Configurations and Micro-configurations

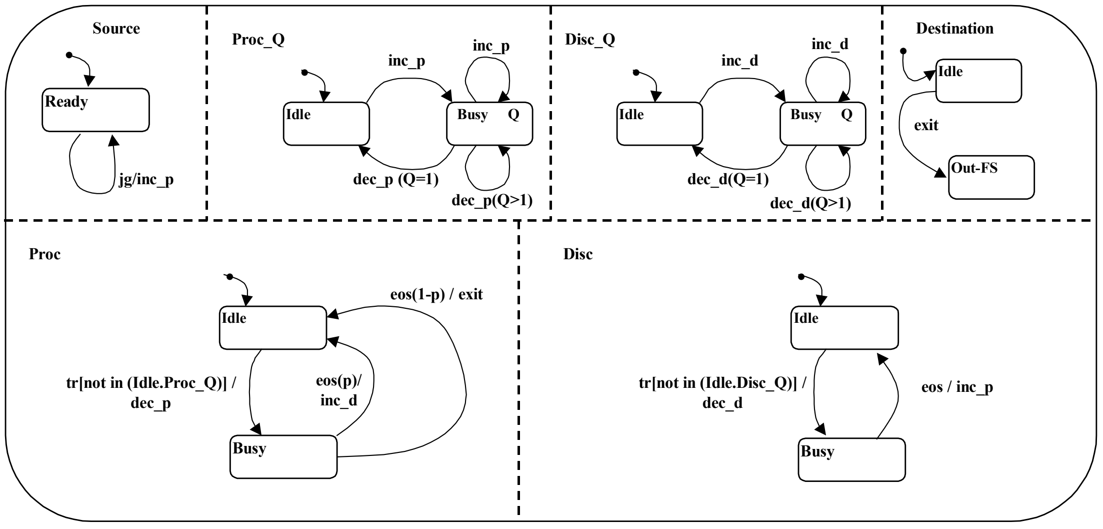
Statecharts semantics have as base a sequence of instants of time {σi} i≥0, that corresponds to the rate of execution of the system. In [Harel et al., 1987], a set of time intervals exists defined for step is I= (σ_i, σ_{i+1}), where each $\Delta\sigma$ (σ_{i+1}-σ_i) represents the elapsed time for determined step *i*. The system will react (to events) at the end of each interval $\sigma_i \rightarrow \sigma_{i+1}$, presenting a new configuration of states.

An external stimulus, in σ_{i+1}, is a triple (Π, θ, ξ), where Π is a set of external primitive events that occur in I_i, θ is the set of external primitive conditions whose values are true in (σ_i, σ_{i+1}), and ξ is a function determined for the external environment, thus, for a variable v, ξ (v) = x if the value of v is x in (σ_i, σ_{i+1}). Thus, a configuration of system associated with the instant σ_{i+1} is a tuple (X, Π, θ, ξ), where X is the maximal configuration of states of the root state, and (Π, θ, ξ) is an external stimulus associated to σ_{i+1}.

A reaction of the system is a pair (ϒ, Π*), where ϒ is the set of transitions called *step*, and Π* is a set of atomic events generated by ϒ. Informally, a step is a set of transitions that can be enabled and are induced for external stimulus. It is important to emphasize that all the constant transitions in a step ϒ are fired simultaneously.

Another definition of a step is viewed as a sequence of micro-steps, which generates intermediate configurations (micro-configurations), where each micro-step is contained in ϒ. In order to clarify the idea of a micro-step, an example of a basic file server [Francês et al., 2001], shown in Figure 3, will be used.

The initial global state, the first configuration SC1, is the set of the initial default states. Admitting that the event *jg* (generation of jobs) is enabled, the action *inc_p* (processor queue is increased) is carried out in the parallel state Proc_Q. Based on this, the next configuration (active basic states of each orthogonal component) is SC2=(Ready, Busy.Proc_Q, Idle.Proc, Idle.Disq_Q, Idle.Disk, Idle.Destination). This particular configuration triggers the so called immediate event (events automatically generated by the internal logic of Statecharts) *tr[not in Idle.Proc_Q]* leading to an intermediate configuration (Ready, Idle.Proc_Q, Busy.Proc, Idle.Disk_Q, Idle.Disk, Idle.Destination). This can be considered as a micro-step generating a micro-configuration (an intermediate configuration). However, this micro-configuration is reacted again and immediately to the action dec_p associated to *tr[not in Idle.Proc_Q]* generating SC3=(Ready, Busy.Proc_Q, Busy.Proc, Idle.Disk_Q, Idle.Disk, Idle.Destination).



Obs: Where Q is the number of customers in a queue

Figure 3. File Server Using Statecharts Representation.

If all the states are assumed to be immediate there would be no inconsistency between configurations and micro-configurations. However, one can come up with a situation where the Busy states of the servers (Proc and Disk) of the model in Figure 3 possess a certain τ_i, that effectively represents values of time and, possibly, different values, which are interpreted as an average of a determined probability distribution (assumed here as exponential). It can still be imagined that, for the example, $\tau_{source} > \tau_{busy.proc} \wedge \tau_{source} > \tau_{busy.disc}$. Thus, in hypothetical values, $\tau_{source} = 5$ u.t., $\tau_{busy.proc} = 3$ u.t. and $\tau_{busy.disk} = 4$ u.t.

Under these circumstances, SC1 would continue to be the initial configuration and after executing event *jg* and the action *inc_p* the next configuration SC2 would be (Ready, Busy.Proc_Q, Idle.Proc, Idle.Disq_Q, Idle.Disk, Idle.Destination). Due to the reaction to the immediate event *tr[not in (Idle_Proc_Q)]* SC3 would be (Ready, Idle.Proc_Q, Busy.Proc, Idle.Disk_Q, Idle.Disk, Idle.Destination). However, configuration SC4 already would generate a structural conflict of which would be the responsible event for going off the micro-steps that compose SC4. If the transition that contains *eos* is enable, then the system will be able to reach two possible configurations SC4: (1) if probability *p* is admitted, then SC4 will be (Ready, Idle.Proc_Q, Idle.Proc, Busy.Disk_Q, Idle.Disk, Idle.Destination); (2) if the probability *1-p* is admitted, then SC4 will be (Ready, Idle.Proc_Q, Idle.Proc, Idle.Disk_Q, Idle.Disk, Out_FS). Moreover, if the qualified transition will be the one that contains the event *jg*, then configuration SC4 would be (Ready, Busy.Proc_Q, Busy.Proc, Idle.Disk_Q, Idle.Disk, Idle.Destination).

One way to maintain consistency for successor configurations is by adopting a timeline and timers in each state that has an associated delay. Thus, the choice would be conditional to the least elapsed time (associated to a state) to qualify for an immediate transition.

3.3. Redefinition of Steps and Configurations to Stochastic Statecharts

This section introduces the idea of time in steps and configurations (dynamics) of Statecharts. As defined in [Harel & Politi., 1998], a system is non-deterministic in a given Statecharts configuration (SC) if it has two possible reactions (Υ_1, Π_1*) and (Υ_2, Π_2*), such that $\Pi_1^* \neq \Pi_2^*$ or $\Upsilon_1 \neq \Upsilon_2$.

Thus, in Stochastic Statecharts, a configuration and a step have one interpretation in case all its states are immediate, and another interpretation if states are associated with delays. An extension of the original definitions is required to reach the temporal features of the Statecharts.

Definition 1a: a state is considered with delay when a variable τi exists, which when evaluated it determines:

- The mean time between arrivals of customers, in case a state is a generating source of these customers (Source). It is assumed that the inter-arrival times are exponentially distributed;
- The mean service time to attend the customers, in case a state is a server (in its Busy state). It is assumed that the service times are exponentially distributed.

Definition 2a: a state is considered immediate when its delay is considered zero (τi = 0).

Definition 3a: The time (σi+1) of the next reachable configuration SC = (X, Π, θ, ξ) obeys the following premises:

- The variable evaluated in ξ is τi, where τi is a delay associated with a state si, with i=1.., n;
- min(τi) is the function that indicates the least amount of time of delay in a configuration SCj, with j=1.., n;
- The time spent in each step is $\tau\Upsilon$j = $\tau\Upsilon$j-1 + min(τi), with j=1.., n. The time $\tau\Upsilon$j is equal to the final time of a configuration SCi or to the initial time of a configuration SCi+1;
- If j=1, first step, then $\tau\Upsilon$j = min(τi), therefore $\tau\Upsilon$j-1 = 0;
- τi.rest = {τi} - min(τi), $\forall$ τi $\neq$ min(τi), where τi.rest is the remainder of time of delay of a state that it surpasses for the next step;
- τtotal = $\sum_{j=1}^{n}$ $\tau\Upsilon$j;
- $\forall${τi} - {min(τi)}, τi :=τi.rest, to end of each step;
- If τi = min(τi), then τi.rest= 0 and τi in the following step starts with value 0.

Definition 4a: a selection for probability takes the system to different reactions (Υ1, Π*1), (Υ2, Π*2), ..., (Υn, Π*n), such that $\Pi^*1 \neq \Pi^*2 \neq ... \neq \Pi^*n$ or $\Upsilon1 \neq \Upsilon2 \neq ... \neq \Upsilon n$.

Definition 5a: if, in a Statecharts, its configurations and times of its steps are determined by definitions 1 to 4, then this Statecharts is defined to be Stochastic.

Using the example of Figure 3 and based on the notation extended for the random vision (Figure 4), with the hypothetical values τsource = 5 u.t., τbusy.proc = 3 u.t. and τbusy.proc = 4 u.t, it is possible to trace a timeline to determine the time of each step, that is the beginning and the end of each configuration. Moreover, the order that the configurations occur can be determined.

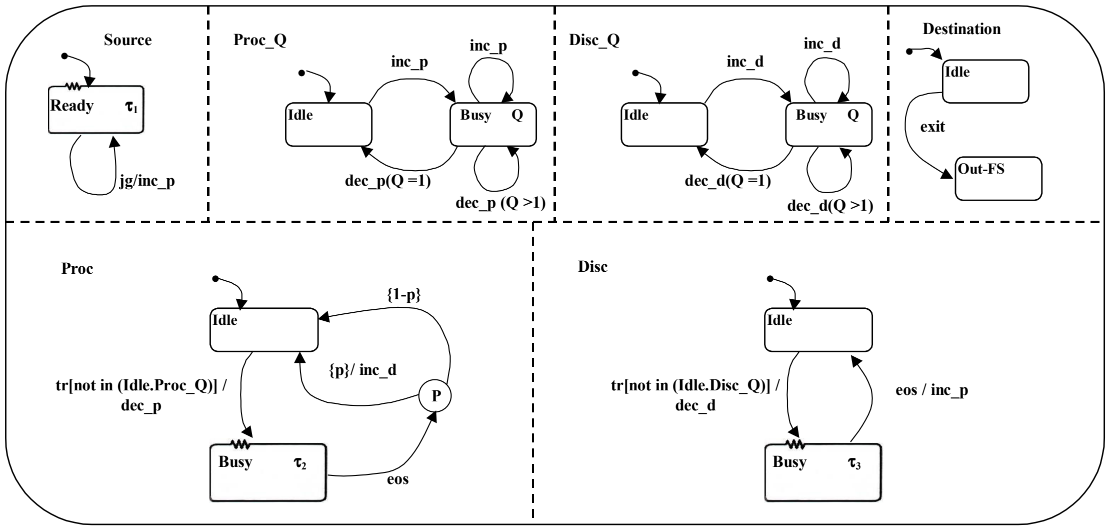


Figure 4. File Server and their States with Associated Delays.

SC1 is the initial configuration, i.e., it presents the entry to the system in the Source state (a state with delay of 5 u.t.), and other initial states of other parallel components. SC2 is the successor configuration, and it is a function of the first one, in relation to the time. If Source state is active, the system waits for 5 u.t. until the first customer is generated (event *jg*). After 5 u.t., step 1 is completed with the execution of the action *inc_p* (in Proc_Q), taking to Busy.Proc_Q. Now, SC3 qualifies for an immediate transition through event *tr[not in(Idle.Proc_Q)]* reaching Busy.Proc, and simultaneously reaches Ready (in Source) through *jg*. The time of the step and the next configuration will be determined by previously described functions (in accordance with the stochastic vision), in the following way (being $\tau_1 = \tau_{source}$, $\tau_2 = \tau_{busy.proc}$, $\tau_3 = \tau_{busy.Disk}$):

- Time of step 1 is $\tau_{\Gamma 1} = \min_{passo1} (\tau_{source}) = 5$, therefore it only has a state with delay (Source);
- $\min_{passo2} (\tau_{source}, \tau_{busy.proc}) = (5, 3) = 3$;
- Time of step 2 is $\tau_{\Gamma 2} = \tau_{\Gamma 1} + \min(\tau_{1,} \tau_2) \Rightarrow \tau_{\Gamma 2} = 5 + \min(5, 3) = 8$, or either, the next step starts in 8 u.t.;
- The time that surpasses the time of step 2 for different states that have the minimum time is $\tau_{i.resto} = \tau_i - \min(\tau_i) \Rightarrow \tau_{1.resto} = \tau_1 - \min(\tau_{1,} \tau_2) = 5 - 3 = 2$, that is, the time that it surpasses step 2 for step 3 is of 2 u.t. in the Source state;
- When step 2 is to complete 8 u.t. (to its ending), then $\forall \{\tau_i\} - \{\min(\tau_i)\}$, $\tau_i = \tau_{i.rest}$, meaning that the states that had not completed their respective delay in one determined step, start the next one with its equal times to the remainder of the previous step. For the example, $\tau_1 = \tau_{i.rest} = 2$. Figure 5 presents the timeline, with the times of each step, in accordance with the gotten values previously.

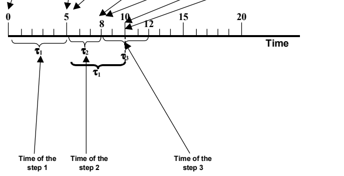


Figure 5. Timeline with Configurations and Steps.

The temporal values for SC3 are calculated in the following way:

- $\min_{step2}$ (τ_{source}, $\tau_{busy.Disk}$) = (2, 4) = 2;
- Time of step 3 is $\tau_{\Gamma 3} = \tau_{\Gamma 2} + \min(\tau_{1,}\ \tau_3) \Rightarrow \tau_{\Gamma 3} = 8 + \min(2, 4) = 10$, that is, the next step starts in 10 u.t.;

The time that surpasses the time of step 3 for different states that have the minimum time is $\tau_{i.rest} = \{\tau_i\} - \min(\tau_i) \Rightarrow \tau_{3.rest} = \tau_3 - \min(\tau_{1,}\ \tau_3) = 4 - 2 = 2$, that is, the time that it surpasses of step 3 for step 4 is of 2 u.t. in the Busy.Disk state. To proceed admitting the previous premises, some considerations must be established. First, it is important to clarify where it finds the random character of the specification and the admitted functions. The randomness is intrinsic to the distributions of generation and attendance to the customer, therefore they are these distributions that generate the rhythms of arrivals and of service (variables that serve of input for the solution of the system). The definite functions as premises warrant that the events can keep the order waited for a generic system of queues, what does not harm in nothing the random approach.

In order to represent queuing systems there is a general agreement to use certain standards to describe states and events. This is exactly the objective of the following section in which some templates are provided to represent features (states and standard events) of queuing systems to be used while specifying a given model in Statecharts.

4. Case study: Modeling of the Scheduling of Processes in a Distributed Environment

This section presents a case study, involving distributed programming and its mechanisms used for scheduling processes, that constitutes an area where modeling can be seen as an attractive aid. More specifically, the experiment carried out involves the allocation of processes of a certain application onto distributed processing elements, using a message passing environment and its respective methods of scheduling. Typical situations of distributed programming and scheduling of processes constitute interesting cases of study, due to the inherent complexity of the problem.

The environment considered for the experiment is composed of five servers. The first one (the PVM server) is responsible for the activation of the processes in the servers lasdix, lasdpc08, lasdpc10 and lasdpc11. Figure 6 is the representation based on Stochastic Statecharts. There is no external source for customer generation as the model is a closed one. All customers are generated by a server embedded in the system (actually PVM generates all the considered 400 parallel processes). The generation of processes takes place when PVM is invoked, considering that it does not have other processes in its queues (event *tr [in (Idle.PVM_Q)]*); PVM returns to the normal course of the application code (event *cg*, in the state *Code*) after receiving a feedback by the application.

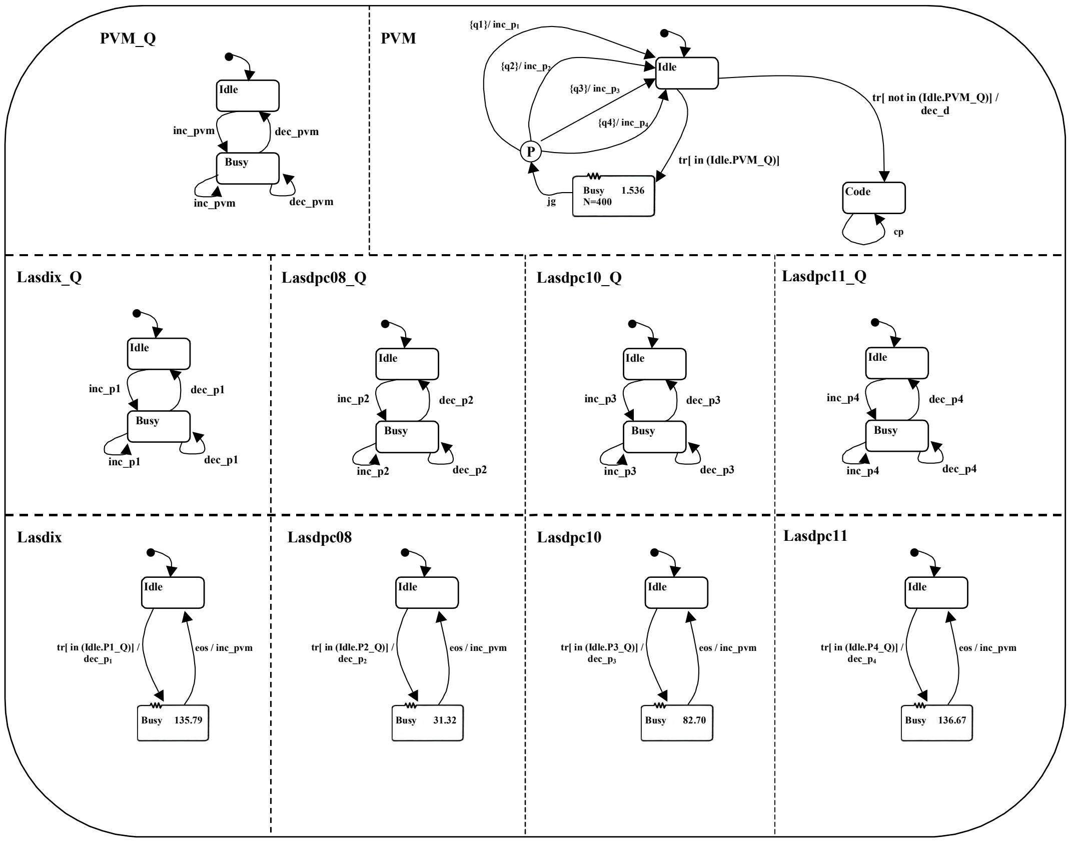


Figure 6. Specification of the PVM Model in Stochastic Statecharts.

In order to illustrate the difference in specification, the same model is represented by using Generalized Stochastic Petri Nets (GSPN) [Chiola et al, 1993] and it is shown in Figure 7.

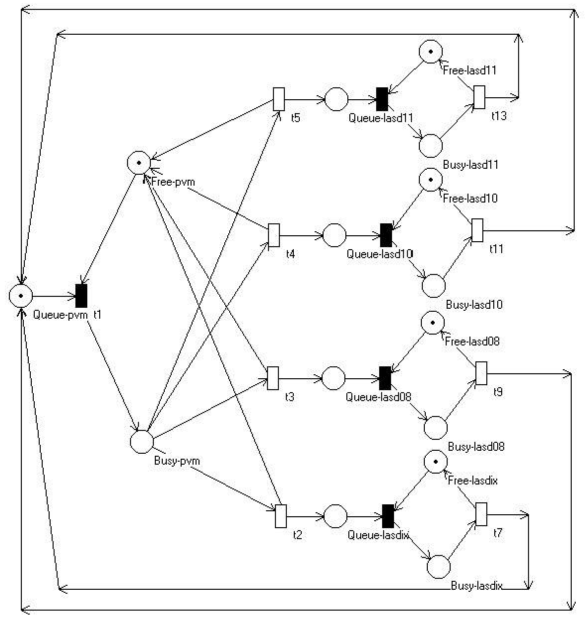


Figure 7. Specification of the PVM Model in GSPN.

The parameters adopted for the model presented in Figure 6 were obtained from the system described in Table I.

TABLE I. Description of the Instrumented Environment.

Machine	Processor	RAM	HD	Network Interface Card	SO
lasdix	Pentium MMX 200MHz	32MB	4GB	TrendNET 10/100	Linux 2.2.16-13cl
lasdpc08	Pentium II 400 MHz	64MB	13GB	TrendNET 10/100	Linux 2.2.16-13cl
lasdpc10	Pentium II MMX 233 MHz	64MB	12GB	TrendNET 10/100	Linux 2.2.16-13cl
lasdpc11	Pentium MMX 200MHz	32MB	3GB	TrendNET 10/100	Linux 2.2.16-13cl

The message passing environment [McBryan, 1994] adopted is the PVM - Parallel Virtual Machine [Beguelin, 1994], which is a software package that allows the creation of a virtual parallel machine on a network of heterogeneous computers, emulating a parallel computer. The version of the PVM used in the experiment is the PVM 3.4.2.

The application comprises a program to sort a vector of 300 thousand positions, using the quick-sort algorithm and is built using the SPMD (Single Program Multiple Data) paradigm, in which a master process sends data to the slave processes, that apply the same program to the data sets received.

In order to obtain the mean service times, the function *gettimeofday* of C language has been used at the beginning and at the end of each service of customers. The PVM server runs on the lasdpc08 machine. The other servers are running on the machines with the same name. One hundred processes were generated to be handled by each machine. These processes were scheduled through the standard round-robin mechanism of PVM. The times collected were clustered into classes and the corresponding histograms were generated. Each sample describes an exponential distribution shifted from the origin. The mean service times (in milliseconds) for each server are shown in Table II.

TABLE II. Mean Service Times.

Server	Mean Times (ms)
PVM	1.53686
lasdix	135.7939
lasdpc08	31.32981
lasdpc10	82.705
lasdpc11	136.6733

The data from Table II shows that PVM presents the lowest time and this is due to the fact that this software basically routes tasks among the processors. Furthermore, processing time values are directly related to the computational power of the processors, which is reasonable. For the first experiment, all the machine were considered in the same way and the probability of each machine being chosen by PVM is the same (25%). The data from Table III shows that similar values were obtained from the simulations (smpl and GSPN) and the analytical solutions. On the other hand, the utilization of the different servers shows that the computer with higher computing power (lasdpc08) was the one that was less used, thus leading to overload other machines.

A new simulation was performed considering an associated performance index for each machine. Thus parameters considered were: lasdix = 1.01 x lasdpc11; lasdpc08 = 4.35 x lasdpc11; lasdpc10 = 1.67 x lasdpc11. From these relations, new weighted probabilities for the powers of machines were established.

From the obtained probabilities, the replications of the simulation for one hundred million units with fifteen smpl seeds were executed again. The results of the weighted simulation and equally probable simulation are synthesized in Table IV. By observing Table IV, it can be inferred that it has a higher use of the available resources in the system. The utilization was more balanced in all the machines. About half of the processes are directed to lasdpc08, due to the fact that this machine has a higher computational power.

TABLE III. Results Obtained for Analytical and Simulation Solutions, where U (Utilization).

Server	U(MVA)	U (smpl)	U (GSPN)
PVM	0.044831	0.0447 ±0,00002837	0.045800 ± 0.008400
lasdix	0.990303	0.9874 ±0,00040794	0.988326 ±0.035238
lasdpc08	0.228479	0.2284 ±0,00005365	0.223423 ± 0.011646
lasdpc10	0.603142	0.6012 ±0,00038534	0.580202 ± 0.024037
lasdpc11	0.996716	0.9942 ±0,00021510	0.991806 ± 0.035343

TABLE IV. Results from the Standard Scheduling (1) and the Alternative Scheduling (2) Simulations.

Server	U (1)	U(2)
PVM	0.0447 ±0,00002837	0.0836 ±0,00522877
lasdix	0.9880 ±0,00040794	0.9378 ±0,10364316
lasdpc08	0.2285 ±0,00005365	0.9536 ±0,00649479
lasdpc10	0.6010 ±0,00038534	0.9738 ±0,00467259
lasdpc11	0.9939 ±0,00021510	0.9229 ±0,00533046

5. Final Remarks

Due to the visual appeal presented by Statecharts in representing complex system behavior, it is quite a temptation to use this graphical technique in modeling reactive systems for evaluating their performance. The very first approach towards representing and dealing with performance models consisted in converting the Statecharts representation into a Markov chain in which an analytical solution was used to generate the performance measurements. This paper described a new approach for system modeling as an extension to Statecharts. The proposed extension comprises Stochastic Statecharts and they follow very closely the specifications proposed by Harel for the standard Statecharts. The idea of time in steps and configurations are introduced.

All developments described in this paper were performed in such a way that the application of both analytical and simulation techniques to solve the models are possible. In order to demonstrate the usefulness of the proposed approach, an example considering a file server system (used as a test bed) and a case study of a distributed application running on a PVM-based virtual computing environment were presented.

The experiments conducted show that the application of the approaches proposed is quite attractive and potentially improves the set of modeling strategies available for many research and application areas involving system performance evaluation.

Furthermore, the approach discussed in this paper extend Statecharts by creating an alternative modeling technique that retains the potential features of synchronization, parallelism and hierarchy as well as embedding stochastic features so that performance evaluation can be conducted on generic specifications.

References

Beguelin, A.. PVM: Parallel Virtual Machine – A User´s Guide and Tutorial for Networked Parallel Computing. The MIT Press, 1994.

Chiola, G., Marsan M. A., Conte, G. Generalized Stochastic Petri Nets: A Definition at the Net Level and Its Implications. IEEE Transactions on Software Engineering, vol. 19, n. 2, p. 89-106, 1993.

Francês, C. R. L., Vijaykumar, N. L., Santana, R. H. C., Santana, M. J., Carvalho, S. V., Abdurahiman, V. The Use of Analytical and Simulation Solutions with Statecharts for Performance Evaluation: A Case Study of a File Server Model. In: 2nd IEEE LATIN-AMERICAN TEST WORKSHOP - LATW2001, Cancun, 2001. Digest of Paper, Cancun, LATW, p.136 – 14, 2001.

Francês, C.R.L.; Oliveria, E.L.; Costa, J.C.W.A.; Santana, M.J.; Santna, R.H.C.; Bruschi, S.M.; Vijaykumar**,** N. L.; Carvalho, S.V. Performance Evaluation based on System Modeling using Statecharts extensions**.** *Simulation Modelling Practice and Theory*. In Press.

Harel, D. Statecharts: a visual formalism for complex systems*. Science of Computer Programming*, v. 8, 231-274, 1987.

Harel, D.; Pnueli, A.; Schmidt, J.; Sherman, R. On formal semantics of Statecharts. *IEEE Symposium on Logic in Computer Science*, Ithaca, USA, 1987

Harel, D.; Namaad, A. The STATEMATE semantics of Statecharts. *ACM Transactions on Software Engineering*, v. 5, n. 4, 293-333, 1996

Harel, D.; Politi, M. Modeling Reactive Systems with Statecharts: The Statemate Approach. McGraw-Hill, New York, USA, 1998

Jain, R. The Art of Computer Systems Performance Analysis – Tecnichniques for Experimental Design, Measurement, Simulation e Modeling. s.l, John Wiley & Sons, Inc, 1991.

Kleinrock, L. *Queuing systems*. Vol. 2: Computer Applications, John Wiley & Sons, 1976.

Mcbryan, O. A. A Overview of Message Passing Environments. Parallel Computing, v. 20, p. 417-444, 1994.

Peterson, J.L. Petri Nets: an Introduction. s.l., Prentice Hall, Inc., 1981.

Vijaykumar, N.L. *Statecharts: Their use in specifying and dealing with Performance Models*. ITA, São José dos Campos, Brasil, 1999, (Ph.D. Thesis).

Vijaykumar, N. L., Carvalho, S.V. & Abdurahiman, V. (2002). On proposing Statecharts to specify Performance Models. *International Transactions in Operational Research (ITOR)*, 9(3), 321-336.

Vijaykumar, N. L.; Carvalho, S.V.; Abdurahiman, V.; Andrade, V.M.B. Introducing probabilities in Statecharts to specify reactive systems for Performance Analysis, *Computers & Operations Research*. In Press