Stochastic Extension to Statecharts for representing performance models: an application to a file system

Carlos Renato Lisboa Francês[1], Nandamudi Lankalapalli Vijaykumar[2]

Marcos José Santana[3], Solon Venâncio de Carvalho[2]

Regina Helena Carlucci Santana[3]

[1] Computing Department-CCI
Universidade da Amazônia-UNAMA
Belém, PA, Brazil
frances@icmc.sc.usp.br

[2] Laboratory of Computing and Applied Mathematics-LAC,
Instituto Nacional de Pesquisas Espaciais-INPE
São José dos Campos, SP, Brazil
{vijay, solon}@lac.inpe.br

[3] Computing Department-ICMC
Universidade de São Paulo-USP
São Carlos, SP, Brazil
{mjs, rcs}@icmc.sc.usp.br

Abstract—

The paper aims to present that another high-level specification tool Statecharts may also be used to represent performance models. Statecharts may be considered as an extension of state-transition diagrams added with the concepts of hierarchy and parallelism. The Statecharts specification has to be treated in the sense performance measurements are to be obtained. This is done by generating Statecharts equivalent of state-transition diagrams. By associating the resulted state-transition diagrams to a Markov chain and using an analytical numerical method, performance measurements are obtained. A performance model represented in queuing networks, Petri nets and Statecharts will be discussed.

Keywords— **Performance Models, Specification tools, Statecharts, Analytical methods, Markov chains**

I. INTRODUCTION

In general, a system behavior may be studied by modeling it and evaluating its performance. It is possible to detect some bottle-necks during its evaluation. Modeling has become an important topic of research as the representation of a given system allows an insight of how the system behaves before it becomes ready. As examples, distributed systems, manufacturing systems and many others may be mentioned.

The scope of this work concentrates on systems that are known as complex as they are reactive. Reactive systems are those that the behavior of the system changes at a given instant according to a perturbation of the system. These type of systems are everywhere in the daily routine such as telephones, cars, communication networks, operating systems, etc.

In order to evaluate the performance of a given system, two categories of solutions are used. The first one is simulation and the other category is analytical methods. The scope of this paper is concentrated on solutions via analytical methods. Usually, the analytical methods are based on Markov theory. It is essential to denote that the main concern of this paper is the modeling of a given system in a clear manner and at the same time it should be possible to be treated computationally.

A very first natural solution of representing a reactive system is by selecting state-transition diagrams. If the events among the states follow an exponential distribution, then this structure may be considered as a Markov chain. Once a system is represented by state-transitions diagrams and by applying appropriate numerical methods [PHI 92] and [SIL 92], one can determine the steady-state probabilities with which performance measurements may be obtained. However, the use of state-transition diagrams may be cumbersome especially when dealing with parallel components as they may lead to exponential blow-up of the model. This problem leads to the necessity of coming up with better representation tools.

There are already widely used tools to cope up with the representation of rather complex systems and these are queuing networks [KLE 76] and Petri nets [PET 81], [MUR 89] and [MAC 96]. Originally, queuing networks have been developed to observe two fundamental factors: queues of requests for a given service and a resource that provides service. Petri nets are also very popular and a very great deal of work has been done using this method. Some extensions to Petri nets, Stochastic Petri nets and Generalized Stochastic Petri nets, are available deal with performance models.

The paper here explores the use of a new approach in adopting Statecharts [HAR 87a], [HAR 87b], [HAR 90] and [HAR 96] to represent performance models. The question is how to solve the specified model in order to obtain the performance measurements. The solution is in generating its equivalent representation in state-transition diagrams that has a one-to-one correspondence with a matrix structure. This matrix is the input to the available numerical methods to determine the steady-state probabilities, basis to yield performance measurements [AND 97].

The idea of adopting Statecharts to represent and treat performance models is another alternative to specify a given system in a high-level of abstraction. This is achieved due to the basic features provided by Statecharts that are extremely useful in depicting most of the necessities (hierarchy, parallelism, entry by history, etc.) essential in representing performance models. In order to show this capability, a prototype was developed [VIJ 99b] and [VIJ 99a].

The next section describes the features of Statecharts. The section III shows the method used to treat the represented system to obtain the performance measurements. Then, section IV is devoted to show an example of a file system [SIL 00] represented in queuing networks, Petri nets and Statecharts. The results obtained are also shown.

II. STATECHARTS

Statecharts, are graphics-oriented capable of specifying reactive systems. This tool is an extension of state-transition diagrams by adding concepts of hierarchy (depth), orthogonality (representation of parallel activities) and interdependence (*broadcast-communication*), i.e., it is an attempt to address both depth and modularity problem as well as the concurrency problem [HAR 87a] and [HAR 87b]. The basic elements that make part of this high-level specification tool are: States, Events, Conditions, Actions, Expressions, Variables, Labels and Transitions. A brief explanation of these elements is followed.

States are used to describe a certain aspect of a given system. Take, for example, a machine that is responsible to process a job. The machine is idle until an arrival of a job takes place. The initial idle stage of the machine may be denoted as a "Waiting" state waiting for a job in order to start the processing. Another state would be "Processing" once a job is fed to be processed and it is the machine's responsibility to respond it for the processing appropriately. For example, the machine may fail while processing a job. In this case one more state "Failure" would represent this fact. The states are shown in Figure 1 and are named as W (Waiting), T (Processing) and F (Failure).

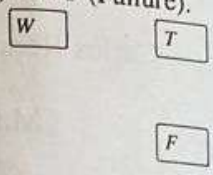

Fig. 1 States

Event is a very important element to observe the model of a given system. It is considered as an interference to the system in the sense that the present system behavior as a whole is changed to another behavior with its (event's) occurrence. From the above example of the machine, *arrival of a job* is an event. The system remains in the "Waiting" state until a job is fed thus changing the system's state from "Waiting" to "Processing" state in which the job is processed. The *end of service* may be considered as another event to denote that the processing is through and that the system should return to "Waiting". In case the machine fails during processing, the event *failure* takes the system to "Failure" state and returning to "Waiting" through the event *end of correction* of the failure.

Optionally, an event may be attached to a *condition* also known as a guarding condition. The Statecharts, in this case, enable the event even though fired only when the condition is satisfied.

Parallelism is used to influence other orthogonal components in the system. This is done by providing the *action* entity. Action may be a change of an expression, a change of a variable or even events that are fired in other orthogonal components. This is associated to the event meaning that once the event is fired, the system's state is changed and the given action is executed so that the system's state is continued to be changed. Statecharts provide some special events such as *true (condition)*, *false (condition)*, *entered(X)* and *exited(X)* (these special conditions are abbreviated in the Statecharts formalism as *tr(condition)*, *fs(condition)*, *en(X)* and *ex(X)* respectively) to cope up with the internal logic of the modeled system. The first two are respectively true and false if the condition is evaluated to be true and false. The last two are related to a state X in which the event *en(X)* is stimulated if state X is entered whereas the event *ex(X)* is stimulated whenever there is an exit from state X. These four special events are considered as immediate events which means that these are the first to be reacted and automatically in any given system.

Statecharts formalism classifies events into two categories: external and internal events. In the case of performance models, it is decided to keep these categories with the following definitions. External events are the stochastic events (where time between their activation and their occurrences follow a stochastic distribution) that have to be externally stimulated to yield new configurations. Internal events are those special (immediate) events mentioned above (*true (condition)*, *false (condition)*, *entered(X)* and *exit(X)*) where they are always checked and continuously reacted until none of them are active. Actions are also considered as internal events. This discussion will be detailed in the next chapter.

Transition is the physical representation to denote a change of a state within a system. Transitions are represented by arrows. A label is provided on the arrows to indicate the events that change the state of a system behavior. By making use of the example of the machine, there would be an arrow from "Waiting" (W) state to "Processing" (T) state. The label of the arrow could be the event *arrival of a job* (a) indicating that a transition from "Waiting" state to "Processing" state has occurred when the event has been fired. This is shown in Figure 2. The full syntax of a label along a transition is: *ev[c]/a*. Event *ev* is fired if the condition *c* is satisfied and when this *ev* is fired, a change of system's state has occurred. Then the action *a* is executed by changing again the system's state (in other components). The Figure 2 also shows other events to indicate *end of service* (s), *failure* (f) as well as *end of correction* (c).

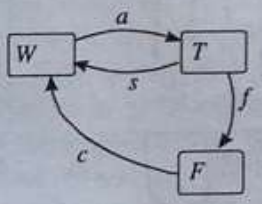

Fig. 2 States, Events and Transitions

The main features of the Statecharts by using the above mentioned entities will be discussed in the following sections. These features provide appropriate resources to specify a reactive system in a clear and realistic manner.

A. *Hierarchy of States.*

In Statecharts, a state may contain sub-states. This is very useful in performance models when events trigger the same transition from different states. Instead of representing a physical transition leaving from each state, just an arrow leaving from the father state also known as a super-state is sufficient. In this case the number of transitions in the diagram is reduced thus simplifying the representation of the model.

Figure 3(a) shows a state-transition diagram representation of a model of a machine to execute two types of jobs. The possible states for the machine are: idle (W), working on job of type 1 (T1), working on job of type 2 (T2) and failure (F). The transitions can be fired by one of the following events: arrival of a job of type 1 (a1) or type 2 (a2), end of job of type 1 (s1) or type 2 (s2), machine failure (f) and end of repair (c). It is assumed that a job is executed only if the machine is idle and, when idle, the machine is not subject to failures.

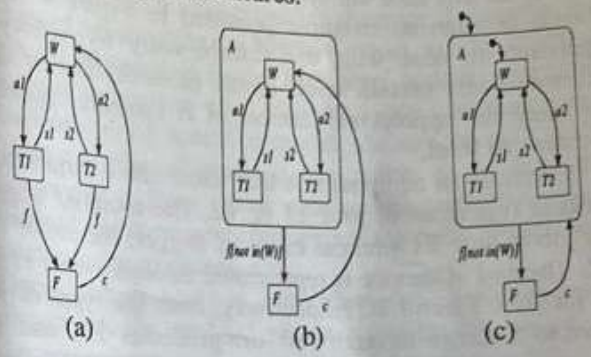

Fig. 3 Example of Hierarchy

In Figure 3(b), the states W, T1 and T2 are clustered in state A (Available). The state A is said to be a macro-state or a super state of type XOR. The semantics of A is interpreted as whenever this state is active, one and only one of its sub-states is active. The event f takes the machine from "available" state to "failure" state F. In order to represent the fact that the machine is not subject to failures when idle, the condition [not in(W)] is added to the transition label from A to F. To be more precise the condition must be correctly written as [not in (A.W)] - (indicating state W within the macro-state A) - to avoid ambiguity whenever state names are repeated. However, here this is not necessary as the figure shows no ambiguity in the names thus making the syntax to be formalized as [not in (W)]. This notation – *macro-state.sub-state* – has been adopted for the proposed work (only when necessary) as no satisfactory notation for avoiding ambiguity has been found in the literature that has been read.

In general it is necessary that some state must be a default or initial state in order to denote that some state is the starting point whenever a system is to be observed for its dynamics. It is very common in Statecharts terminology to use "a system is entered" meaning that a macro state is switched on or active. In order to represent an initial state, Statecharts provide the concept of *entry by default*. In Figure 3(c), the bent arrow pointing to the state W means that this state is the initial point of entry within state A. When the system enters in A, one of its sub-states must be active, in this case W is activated due to *entry by default*.

In Figure 3(c), when the A macro-state is active, as mentioned earlier, only one of its sub-states is active. In Statecharts the active states of a given system are referred to as configuration. Therefore, one can say that the initial configuration of the macro-state A is W.

Another way to enter a system is through its *history*, i.e. when a system is entered the state most recently visited is activated. In order to indicate that history is to be used instead of entry by default, the symbol H is provided. It is also possible to use the history all the way down to the lowest level as defined in the Statecharts formalism [Harel, 1987a]. In this case the symbol H* is used. This feature is shown through an example provided in Figure 4. Other approach to deal with the feature entry by history to influence only certain levels of the hierarchy can be done by using the appropriate number of H symbols applied to the desired level.

Consider an equipment A that when idle is requested to process two types of jobs T1 or T2. The event *a1* triggers the job of type T1 whereas event *a2* triggers the job of type T2. The end of service is represented through events *s1* and *s2* for jobs T1 and T2 respectively. Both the types of jobs have to undergo through two sub-processes T11 and T12 (in case of T1) and T21 and T22 (in case of T2). While the equipment is busy, it may fail and this takes to another state F (Failure) through the event *f*. Note that the condition *not in(W)* is attached to the event to guarantee that the equipment may fail only when not idle.

When the failure is corrected the last state visited becomes active and it is represented via the history symbol H. In Figure 4(a) when the super-state A is entered, most recently visited state (W, T1 or T2) will become active bypassing the provided default state. In case T1 is the most recently visited state, the sub-state T11 will become active as it is the default state within T1 while in case of T2, sub-state T21 (default state within T2) will become active. In Figure 4(b) as H* is used the state to be active within T2, for example, may be T21 or T22 (whichever was the most recently visited state).

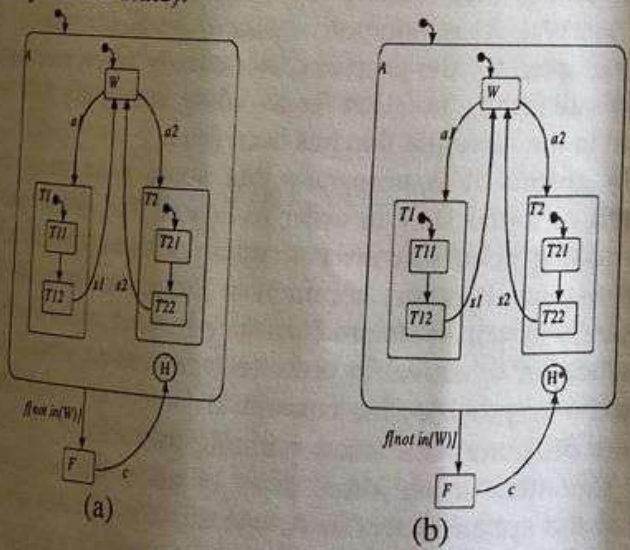

Fig. 4 Entry by History

B. Orthogonality and Interdependence.

Figure 3 showed an example in which the system was sequential, i.e., in each level of hierarchy, one and only one state could be active. However, the performance models have to describe frequently parallel subsystems. The idea of parallelism is expressed in Statecharts by using orthogonal states. This is obtained by defining macro-states of *AND* type. Whenever a macro-state of AND type is active, all its sub-states are also active.

In order to illustrate the idea of orthogonality, consider a system with two machines M1 and M2 (with the same features shown in Figure 3) working in parallel. It is assumed that only one server is available to repair the machines and machine M1 has priority to be repaired over M2. The Statecharts representation of this model is shown in Figure 5. Here, the state of the system is described by the state of machine M1 *and* the state of machine M2. The global state S consists of two sub-states. The set of its sub-states becomes a cartesian (orthogonal) product of the states of its sub-states. Taking the Figure 5, the initial configuration is (W, W) which are the initial default states in the M1 and M2 components. In the figure the notation for the conditioned event *c* in the M2 component makes it clear that "M1.F" has to be used to make it sure that it is the state F within the M1 component is the one being indicated. This event with the condition is used to provide priority to M1.

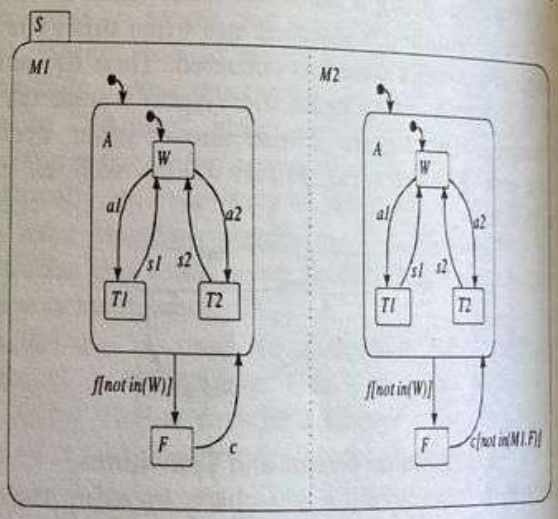

Fig. 5 Orthogonal States

A powerful utility in Statecharts is the possibility of running a broadcast communication. The output events known as actions can be attached (by a "/") to the event to be triggered in a transition. This attached action affects the Statecharts behavior in its orthogonal components.

The capability of representing interdependence is shown in Figure 6. Consider a system with two components M and S denoting respectively equipment and a repairer. The equipment, as in other examples, is responsible to process two types of jobs T1 or T2. The equipment is subject to failure while busy and it will be corrected by the repairer when it is available. The immediate event *true[in(M.F)]* guarantees synchronization in the sense that when the equipment M is in the F state and if the repairer is available

then only a repair takes place. Other events guarantee independence. For example, if event *a1* occurs only the M component is affected when it is idle.

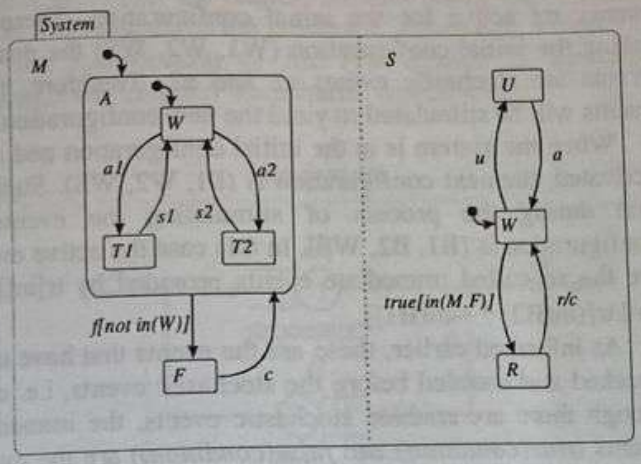

Fig. 6 Synchronization and Independence

C. *Entry by Condition.*

Statecharts tool provides facilities for more complicated entries to sub-states other than Entry by History. One of them is Entry by Condition represented by symbol c. This feature is shown through Figure 7 in which the event to be taken depends on the condition that is to be satisfied. The figure represents a Machine (T) that processes a job either on T1 or T2 according to the conditioned event *a*. Figure 7(a) shows this aspect. By using the connector of Entry by Condition, the number of arcs may be reduced and this is shown in Figure 7(b). This means that even the event occurs, the condition attached is to be evaluated to move to the destination state. In the Figure below, on occurring the event *a* , the conditions c1and c2 are evaluated. The destination state would be selected based on which condition (c1 or c2) is satisfied. Obviously, if no condition is satisfied the transition (through event a) will not be triggered.

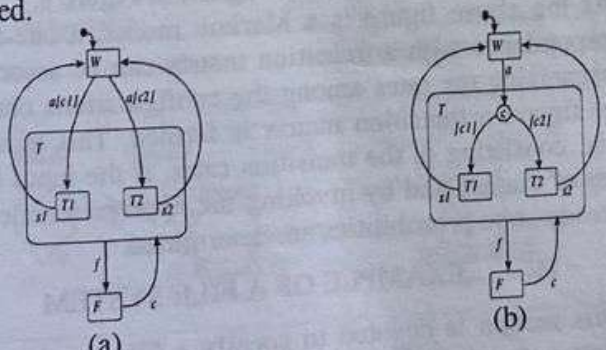

Fig.7 Entry by Condition

III. CONSTRUCTION OF A CONTINUOUS-TIME MARKOV CHAIN FROM A STATECHARTS MODEL

A Markov Chain – more precisely, within the scope of this work, a Continuous-Time Markov Chain - consisting of transition rates among states is the input to the available numerical methods [PHI 92] and [SIL 92] to determine the steady-state probabilities.

It has been already mentioned that a one-to-one correspondence can be made between a Continuous-Time Markov Chain and a state-transition diagram. Therefore, the problem of constructing the Markov model will be solved if the model represented in the high-level specification tool Statecharts generates the state-transition diagram that corresponds to the specified model. As can be seen, with the state-transition diagram, it is possible to produce the transition matrix from which steady-state probabilities are determined and these probabilities are the basis to calculate the performance measurements.

Once the model is specified in a Statecharts representation, the enabled events must be stimulated so that new configurations are yielded. Internal events are generated by Statecharts semantics and considered as immediate events in the sense they are immediately stimulated when the model reaches a configuration where they are active. Events can also be triggered by actions associated to transitions and they (actions) are stimulated as soon as the transition is fired. External events are events that are not generated by the Statecharts semantics and therefore they must be stimulated for describing the dynamics of the modeled system behavior. In order to make the association of Statecharts model with a Markov chain possible, the only type of external events that can be considered are stochastic events. These events are those that the time between their activation and their occurrence follows a stochastic distribution. In particular, for Continuous-Time Markov Chains, this distribution has to be exponential distribution.

Once the model is specified in Statecharts representation, the first step is to take the initial configuration provided by entries by default. In other words all the initial States of each orthogonal component are taken. A reaction, that is stimulation of events to yield a new configuration, is first done by stimulating the so called immediate events such as *true(condition)* and *false(condition)*. This reaction results in a new configuration. If there are actions associated with these immediate events, these are fired yielding new configuration. This process of reacting to immediate events continuously takes place until a configuration is obtained and it is such that no more immediate events are active. Now, a list of stochastic events is obtained based on the resulted configuration. New configuration is generated reacting to each stochastic event of the list. Actions are executed if they are associated with these stochastic events. This new configuration is checked against any active immediate events. This process goes on until all the

configurations have been expanded. The results of all this process is a list of a structure that contains a source configuration, stochastic event (along with its rate) stimulated and the target configuration. With this information a transition matrix has to be generated with which steady-state probabilities can be determined.

The research work proposed here is concentrated on studying this problem and developing a prototype of the mentioned framework. Restrictions (not taking into consideration all the entities of the Statecharts) have been made to tailor the tool to be used for the proposed objective of the framework. These restrictions will be mentioned in the next chapter that describes the implementation details. However, it is felt that for applications in performance models these restrictions do not hinder in any way the specification of a reactive system.

These configurations have to be organized in such a way that a transition matrix consisting of transition rates can be generated, i.e., the rows and columns of the matrix are configurations and the elements correspond to transition rates from a source configuration to a destination configuration. This will be the input for the existing library to calculate the steady-state probability vector and consequently the performance measurements of interest.

In order to understand the complete functioning of the process from the specification until the generation of the transition matrix an example will be shown. Consider a system with three parallel components that correspond to two machinery equipment and a supervisor to repair any failure in the equipment. The components E1 and E2 denote the equipment whereas the component Supervisor is responsible for repairing the equipment when they fail and a priority is provided to repair E1 whenever both the equipment are down. The list of stochastic events include a1, r1, f1, s1 a2, r2, f2 and s2. Internal events are tr[in(B1)], tr[in(B2) ^ ¬in(B1)]. Actions c1 and c2 that are fired once the events s1 and s2 are taken are also considered as internal events. This model is shown in Figure 8.

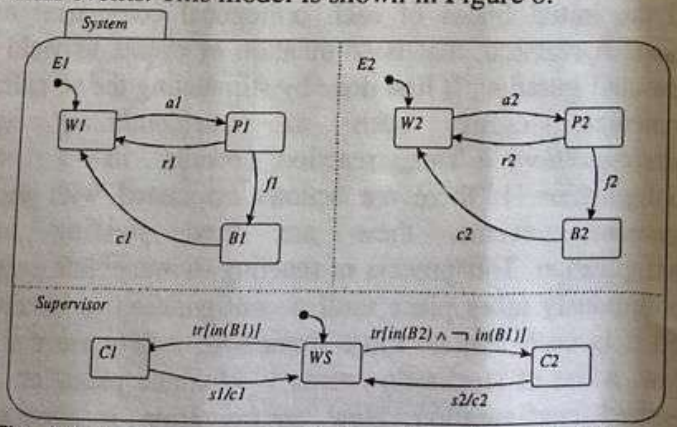

Fig. 8 Statecharts representation of equipment with a repairer

Once the specification is over, a first reaction is performed by first checking the internal events that may be enabled according to the initial configuration and then stimulating these enabled events. For example, internal events such as tr[in(X)], where X is a State, are reacted if the system's initial configuration has an active State X. In the case of the example presented in Figure 8 no internal events are active for the initial configuration. Therefore, taking the initial configuration (W1, W2, WS) the enabled events are stochastic events a1 and a2. Therefore, these events will be stimulated to yield the new configurations.

When the system is in the initial configuration and a1 is activated, the next configuration is (P1, W2, WS). Suppose that during the process of stimulating the events, a configuration is (B1, B2, WS). In this case the active events are the so called immediate events provided by tr[in(B1)] and tr[(in(B2) ^ ¬in(B1)].

As informed earlier, these are the events that have to be checked and enabled before the stochastic events, i.e. even though there are enabled stochastic events, the immediate events (*true(condition)* and *false(condition)*) are the events that have to be stimulated.

The state-transition diagram is shown in Figure 9. As one can notice, this diagram consists of only stochastic events that follow exponential distribution. The states and arcs with events following exponential distribution compose a Markov chain with which numerical methods can be applied to determine steady-state probabilities, the basis for calculating performance measurements.

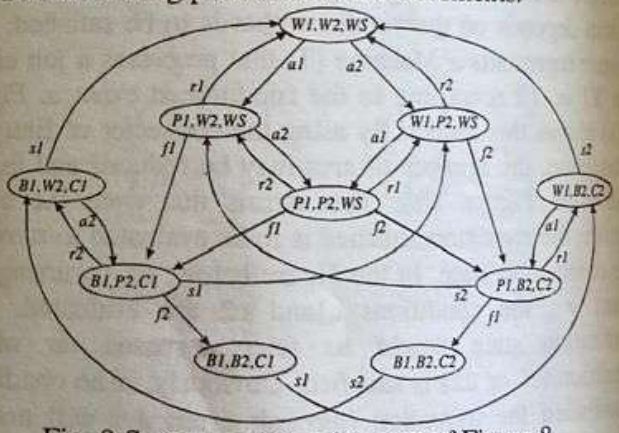

Fig. 9 State-transition diagram of Figure 8.

As the above figure is a Markov model, a one-to-one correspondence with a transition matrix can be associated. By organizing the rates among the configurations obtained in the figure, a transition matrix is formed. This transition matrix, consisting of the transition rates, is the input to the library of classes and by invoking the appropriate methods the steady-state probabilities are determined.

IV. EXAMPLE OF A FILE SYSTEM

This section is devoted to specify a File system using queuing networks, Petri nets and Statecharts. Results of performance measurements obtained by these three representations are also shown.

The file system consists of a processor. Whenever the processor is busy, a request to use the processor is put in a

processor queue. After a request has been processed, it may leave the system with a probability of 0.6 or it may use the disc with a probability of 0.4. The disc also makes use of a queue, disc queue, for storing the requests whenever the disc is in use. After the request finishes occupying the disc service, it is returned to the processor. Figures 10, 11and 12 show the specification of the File System in queuing networks, Petri nets and Statecharts respectively.

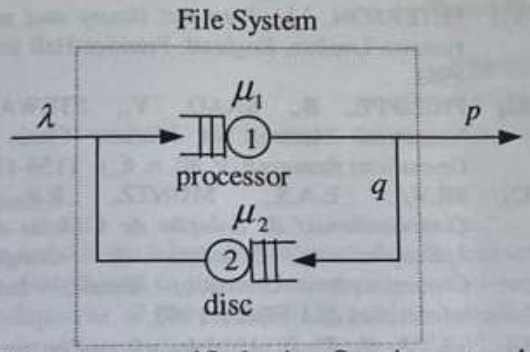

Fig. 10 File system specified using Queuing networks

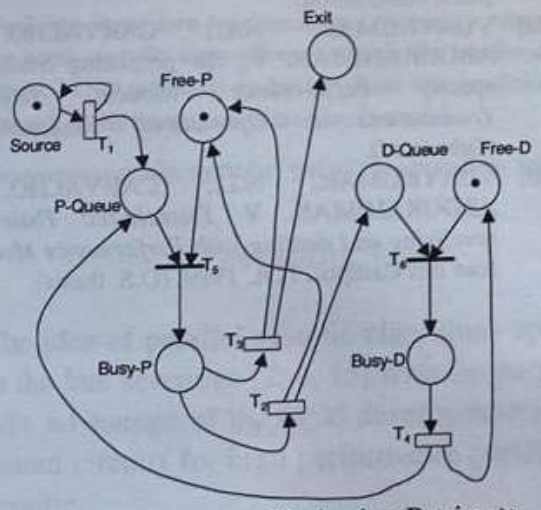

Fig. 11 File system specified using Petri nets

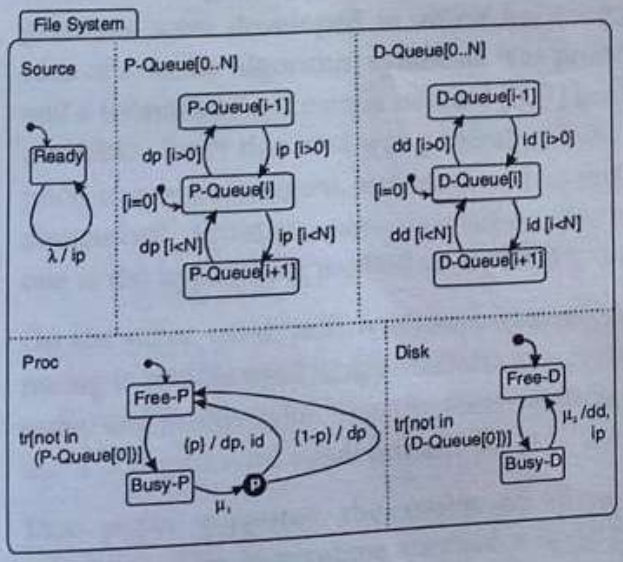

Fig. 12 File system specified using Statecharts

The input data used to produce the performance measurements are shown in Table I.

TABLE I
INPUT DATA FOR THE SPECIFIED SYSTEM

Arrival rate λ	12
Processing rate μ_1	25
Disc rate μ_2	20
Probability of leaving the system p	0.6
Probability of using the Disc q = 1-p	0.4

The results of performance measurements obtained for each specification are shown in Table II.

TABLE II
PERFORMANCE MEASUREMENTS

States	Queuing networks	Petri nets	Statecharts
Processor Busy	0.800	0.799	0.800
Processor Idle	0.200	0.200	0.199
Disc Busy	0.400	0.401	0.400
Disc Idle	0.600	0.598	0.599

The results of performance measurements using queuing networks were based on Jackson networks whereas Petri net solution were derived from simulation. In case of Statecharts, as inclusion of probabilities among the transitions has not yet been added to the prototype a specific program was used. The buffer limits to store the requests in both the Processor and Disc queues were considered unlimited in the case of Queuing networks and Petri nets. However, this is not possible yet using Statecharts. Therefore, the results obtained using Statecharts specification in Table II used the buffer limits as 40 for the Processor queue and 20 for the Disc queue.

By changing the buffer limits other interesting limits were obtained for Statecharts specification. These results are shown in Table III.

TABLE III
PERFORMANCE MEASUREMENTS WITH BUFFER LIMITS FOR STATECHARTS

Processor Queue	5	10	20	30
Disc Queue	**3**	**5**	**10**	**15**
Processor Busy	0.728	0.781	0.798	0.800
Disc Busy	0.384	0.397	0.400	0.400
Processor Idle	0.271	0.218	0.201	0.199
Disc Idle	0.615	0.602	0.599	0.599

The Table III shows that the Processor remains busy by increasing the buffer size for storing the processor requests in the Processor queue. However, this is not true. The

results shown in Table II are based on the buffer size limits of 40 and 20 for Processor and Disc queues respectively. The gain in the Processor usage is not very significant even after increasing the size of the buffer, i.e., once the buffer size for the Processor queue reaches 30, it is not advantageous to increase its size as it will not increase the usage of the Processor.

V. COMMENTS

This paper showed another alternative of representing performance models. Specification of complex systems is not an easy task and it is not the intention of this paper to claim that Statecharts is the best option. However, the potential features provided in Statecharts open a wide range of intricacies to be specified in the model. One more interesting factor is that Statecharts is an extension of state-transition diagrams that are very closely associated to Markov chains which are very much used in analytical solutions to determine the performance measurements.

The prototype has to be improved, in particular, the user interface must be developed as the present status requires the user to write the code in a programming language.

The inclusion of probabilities to the prototype is under development and future work is to study the feasibility of representing Markov Decision Processes.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the valuable help of Ms. Valeria Maria Barros de Andrade for sparing her time and producing the output results of the model by writing a program for Statecharts specification.

REFERENCES

[AND 97] ANDRADE, V.M.B., CARVALHO, S.V., VIJAYKUMAR, N.L Desenvolvimento de um Software para análise de desempenho de sistemas através de modelos markovianos. In: SIMPÓSIO BRASILEIRO DE PESQUISA OPERACIONAL, 29, 1997, Salvador, Brasil. **Anais dos Resumos...** Salvador, Brasil: SOBRAPO, 1997.

[HAR 87a] HAREL, D. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, v. 8, p. 231-274, 1987.

[HAR 87b] HAREL, D., PNUELI, A., SCHMIDT, J., SHERMAN, R. On the formal semantics of Statecharts. In: IEEE SYMPOSIUM ON LOGIC IN COMPUTER SCIENCE, 1987, Ithaca. **Proceedings...** Ithaca, USA: [s.n], 1987.

[HAR 90] HAREL, D., LACHOVER, H., NAAMAD, A., PNEULI, A., POLITI, M., SHERMAN, R., SHTULL-TRAUTIN, A., TRAKHTENBROT, M. STATEMATE: A Working Environment for the Development of Complex Reactive Systems. *IEEE Transactions on Software Engineering*, v. 16, n. 4, p. 403-414, Apr. 1990.

[HAR 96] HAREL, D., NAAMAD, A. The STATEMATE Semantics of Statecharts. *ACM Transactions on Software Engineering*, v. 5, n. 4, p. 293-333, Oct. 1996.

[KLE 76] KLEINROCK, L. *Queueing Systems*. v. 2. New York, USA: John Wiley & Sons, 1976.

[MAC96] MACIEL, P.R.M., LINS, R.D., CUNHA, P.R.F. *Introdução às Redes de Petri e Aplicações*., Campinas, Brasil: Instituto de Computação, UNICAMP, 1996.

[MUR 89] MURATA, T. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, v. 77, n. 4, p. 541-580, 1989.

[PET 81] PETERSON, J.L. *Petri net theory and modeling of systems*. London, England: Prentice-Hall International, 1981.

[PHI 92] PHILIPPE, B., SAAD, Y., STEWART, W.J. Numerical Methods in Markov Chain modeling. *Operations Research*. v. 40, n. 6, p. 1156-1179, 1992.

[SIL 92] SILVA, E.A.S., MUNTZ, R.R. *Métodos Computacionais de Solução de Cadeias de Markov: Aplicações a Sistemas de Computação e Comunicação*. Gramado, Brasil:. Instituto de Informática da UFRGS, 1992.

[SIL 00] SILVA, A R. F. *Modelos de redes de filas para Sistemas Computacionais Distribuídos-Simulação X Métodos Analíticos*. São Carlos, ICMC-USP, 2000, (M.S. dissertation).

[VIJ 99a] VIJAYKUMAR, N.L., CARVALHO, S.V., ABDURAHIMAN, V. On proposing Statecharts to specify Performance Models. *International Transactions in Operational Research*, 1999. *(Submitted)*.

[VIJ 99b] VIJAYKUMAR, N.L., CARVALHO, S.V., ABDURAHIMAN, V. *Statecharts: Their use in specifying and dealing with Performance Models*. São José dos Campos, ITA, 1999, (D.S. thesis).