Proceedings of *I Workshop on Performance of Computational Systems (WPerformance)/Brazilian Computing Conference (SBC)*, Florianópolis, SC, Brazil, July 2002

On embedding memory in Markov Models specified in Statecharts

Nandamudi Lankalapalli Vijaykumar[1], Solon Venâncio de Carvalho[1], Valéria Maria Barros de Andrade[1], Vakulathil Abdurahiman[2]

[1]Laboratório Associado de Computação e Matemática Aplicada (LAC) – Instituto Nacional de Pesquisas Espaciais (INPE)
Caixa Postal 515 – 12201-970 – São José dos Campos – SP – Brasil

[2]Divisão de Ciências de Computação (IEC)
Instituto Tecnológico de Aeronáutica, SJCampos, SP, Brasil

{vijay, solon, valeria}@lac.inpe.br, rahiman@comp.ita.br

Abstract. *Statecharts have been developed to represent reactive systems also known as complex systems. In fact, Statecharts are an extension of state-transition diagrams with notions of hierarchy (depth), orthogonality (parallel activities) and interdependence (broadcast communication). Recently, Statecharts have been adapted to represent and deal (analytically) with performance models. There is an interesting feature in Statecharts that can "remember" the system's state which can not be introduced in Markov theory in a straightforward manner due to its "memory-less" property. This paper proposes a procedure that allows the inclusion of this feature, very useful in performance models, to systems represented in Statecharts. An example of a manufacturing system depicting this feature will be shown.*

1. Introduction

In many a system it is often necessary that decision makers must be able to predict the system saturation so that it is possible to determine the best cost effective means to avoid this situation (or at least delaying this saturation as much as possible). It is interesting and at the same time very important if one has some notion of the behavior of the system both in the best and the worst cases. Thus, typically, management must be in a position to answer some questions, such as *At which point the capacity of the system will be saturated?* and *How the cost can be associated to the failure of the system?*.

One easy way is to depend on the knowledge and expertise of the system administrator or use performance models that can infer from the actual system to tell much more than the "wait-and-see" method [Menascé & Almeida 1998]. Many techniques are mentioned in the literature, and they are generally associated to stochastic processes as these are mathematically well defined and at the same time computationally handled.

A so called low-level and popular technique is the state-transition diagrams as they are very natural to represent performance models as states and transitions, triggered by stimuli or events, among states. On the other hand, higher level techniques such as Queuing networks [Kleinrock 1976], Generalized Stochastic Petri Nets [Chiola et al

1993] and Statecharts [Harel 1987] are to be mentioned to clearly represent the internal logic of such models.

The paper discussed here will use the Statecharts approach to represent [Carvalho et al 1991] and deal with performance models [Vijaykumar 1999], [Vijaykumar et al 2002], [Vijaykumar et al 2000a], [Vijaykumar et al 2000b], [Vijaykumar et al 2001] and [Francês et al 2000]. Performance models have been used for providing the behavior of a system to understand, a priori, about it without the necessity of implementing it. In this context, some metrics can be very interesting for determining performance issues, for instance, response time, throughput, utilization, cost, etc.

Now, the problem faced is not only to represent the model but it is also necessary to solve it so that performance measurements of interest are yielded. In this regard a solution, taking the analytical approach, would be to associate a mathematical method to the specification technique in order to perform the desired analysis. This is exactly the logic that is utilized when Statecharts represent and deal with performance models. However, in many performance models it is usual to face a situation where, after a component is turned off, a return to “the last active state” of the system is necessary. As an example, consider a model in which a robot is responsible to do some service. In case there is a robot breakdown, it has to be repaired and it should return to the state where it was before breakdown has occurred to finish the interrupted task. This is easily represented in Statecharts as will be shown in the section that describes some of the main features. The problem is to solve the model using Markov chain. Due to the memory-less property of Markov chains – future depends only on the present and not on the past – the problem of dealing with this situation need to be handled during the model’s construction. Thus, this paper aims at discussing how memory can be automatically eliminated from Statecharts models in order to obtain a Markov graph. Examples of manufacturing systems that use this feature are given.

The next section briefly introduces Statecharts and just the memory feature is described. Following this, the procedure of handling Statecharts representation to obtain performance measurements is presented. Then, the solution of how to deal with the feature of memory used in models represented in Statecharts is provided followed by an example. Finally some conclusions are commented. Some projects leading to future work related to this area of performance models are also briefed.

2. Statecharts

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 and 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 an 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 further 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].

By definition, when modeling a given system, there must always be an initial state also known as default state in Statecharts. This is the entry point of the system. 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 1. 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.

In order to illustrate this feature, consider an equipment A that when idle can be 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 1(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 1(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). 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.

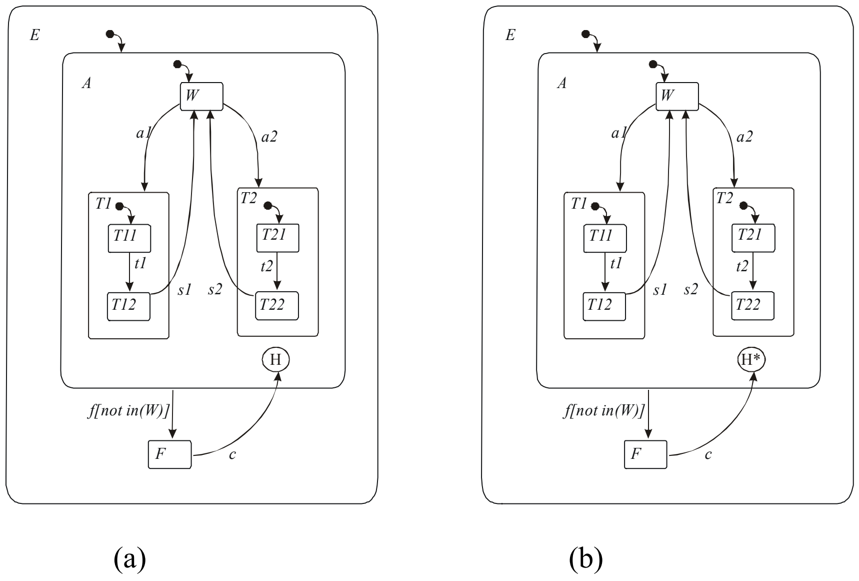

Figure 1. – Entry by History

For a more detailed description of the Statecharts formalism, the readers are requested to refer to [Harel 1987a], [Harel et al 1987], [Harel & Naamad 1996] and [Harel & Politi 1998].

3. Construction of a Continuous-Time Markov chain from a Statetcharts 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 [Philippe et al 1992] and [Silva and Muntz 1992] to determine the steady-state probabilities.

It is a fact 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 Statecharts generates the corresponding state-transition diagram.

Once a model is represented in Statecharts, the enabled events must be stimulated so that new configurations (set of active states of all the parallel components) are yielded. Internal events (*true(condition)*, *false(condition)* and actions) are automatically stimulated by Statecharts semantics as they are considered as immediate events. External events are events that are not generated by the Statecharts semantics and therefore they must be explicitly 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.

In order to understand the complete functioning of the process from the specification until the generation of the state-transition diagram 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 2.

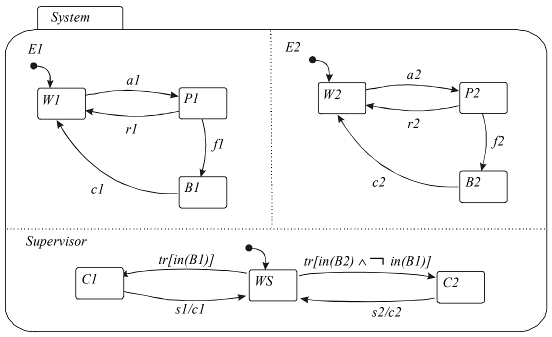

Figure 2. Statecharts representation of equipment with a repairer

Once the specification is over, a reaction is performed by first checking the internal events that may be enabled according to the initial configuration and then stimulating these enabled events. In the case of the example presented in Figure 2 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 3. 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.

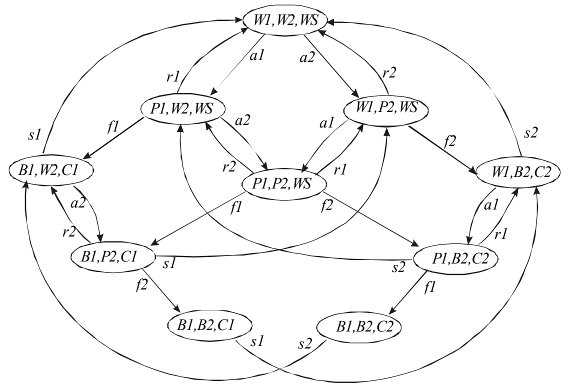

Figure 3. State-transition diagram of Figure 2.

Now, with the Markov chain, 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 that implemented numerical methods to determine the steady-state probabilities.

4. Memory in Markov models represented in Statecharts

One powerful feature, provided in Statecharts is the Entry by History. It is very useful in specifying performance models [Carvalho 1991] and [Carvalho et al 1991]. However, due to the memoryless property in the Markov chains, - future behavior of the process does not depend on the past behavior given the present state of the chain - Entry by History seems, at a first glance, not to be compatible with Markov models as it has to depend on the past information. But, in fact, memory is frequently introduced in Markov models just by adding the necessary past information into the state space as an additional component. Due to the visual appeal of Statecharts and its Entry by History feature, representation of performance models can be very much enhanced. Therefore, a solution to automatically cope up with this feature is provided. In [Vijaykumar 1999], a solution is presented to deal with Entry by History in Markov models. The main idea consists of adding the history information in each possible "present" state configuration by creating a new orthogonal component to the root state.

In order to illustrate this idea, observe Figure 4 where the history feature is depicted. The Figure illustrates an equipment that accepts requests for three types of services T1, T2 or T3. Event a1 takes the equipment to process service T1, event a2 takes the equipment to process service of type T2 and event a3 service of type T3. The equipment may fail but it is assumed that the failure occurs only while processing. Event f takes the equipment to a failure state F. Once it is repaired it returns to the last active service (T1 or T2 or T3). The figure shows this representation of the equipment that when returning from F state to P macro-state, the state active within the P macro-state has to be the last visited state among T1, T2 and T3. In order to cope up with this situation, somehow the solution to be proposed has to "memorize" the last active state among T1, T2 or T3 before a failure occurs so that this active state is reached after returning from failure (F state). The solution that is proposed here is to create another dummy component, named for example, History(P) consisting of the states that are essential to history plus one more state Active. This extra state is used whenever the P component is active. Whenever the event f takes the system from P macro-state to F state (failure state) the state in which the P component was before occurring a failure is "memorized" by making the corresponding state in the dummy component active.

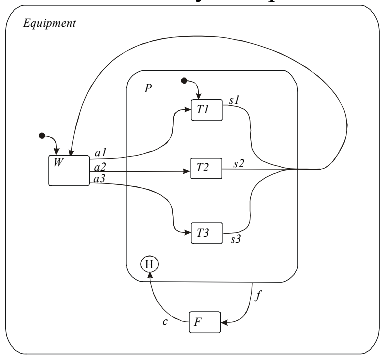

Figure 4. Entry by History feature in a model

One can notice that in the dummy component, the solution made use of events already defined in the Statecharts formalism such as *en(X)* and *ex(X)* that respectively mean that they are stimulated each time X state is entered and when X state is exited. This situation is shown in Figure 5. This means that in case P macro-state is left while in T1, immediately the dummy component History(P) would make the HT1 sub-state active as the event *ex(P) ^ ex(T1)* becomes true. As one can notice regardless of the destination state from P (either F or W), T1 is "memorized". The "memory" aspect will be used only when returning from F state.

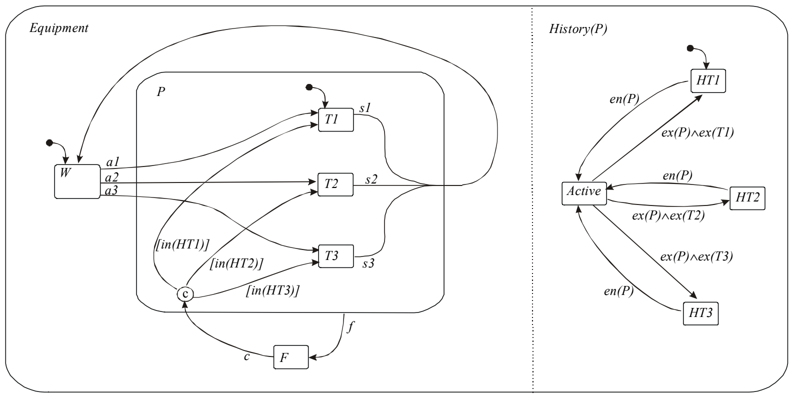

Figure 5. A solution to Entry by History

Note that in the proposed solution, a dummy state History(E) must be created for each state E that contains Entry by History. In the general case, these dummy states must be the direct offspring of the root state i.e. they must be on the same level of the other orthogonal components of the root state. History is eliminated using this solution, i.e. past information becomes part of the present configuration. A state-transition diagram of the above figure is presented in Figure 6.

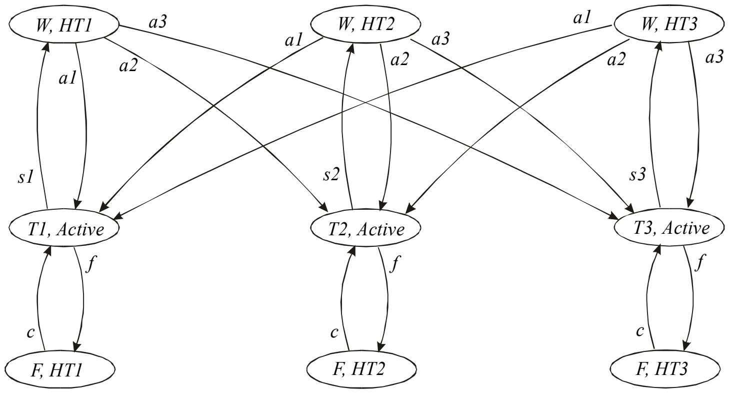

Figure 6. State-transition diagram of Figure 5.

Just for illustration purposes of using entry by history all the way down within an hierarchy, example of Figure 1 is considered. Figure 7 shows the addition of another orthogonal component in order to deal with H of Figure 1a. As one can notice, just the macro-states T1 and T2 are necessary to be remembered due to the use H symbol in Figure 1a. Once T1 or T2 is active, their default states (T11 or T21) will be active anyway. Therefore, "remembering" T1 or T2 is enough.

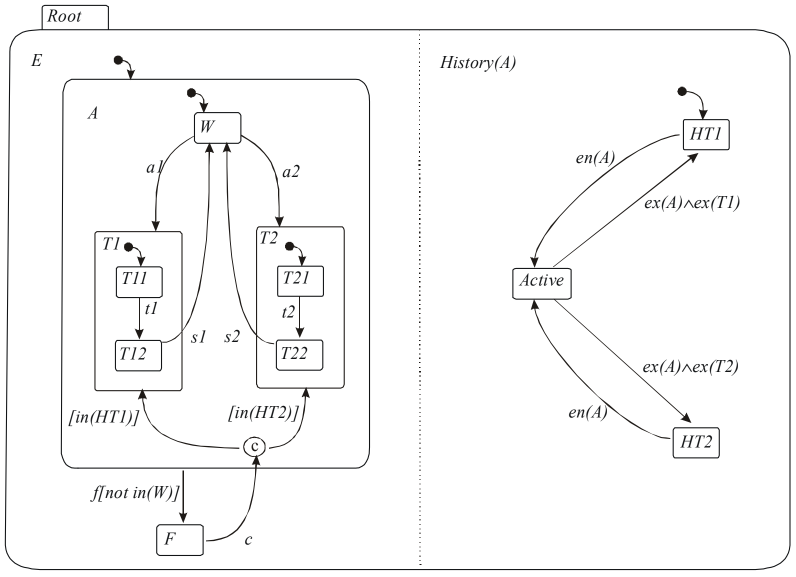

Figure 7. Dealing with History of Figure 1a

Now, Figure 8 shows the addition of the orthogonal component to deal with Figure 1b where H* is used and this symbol denotes to go all the way down to the lowest level. In this case it is necessary to "remember" all the sub-states within T1 and T2. Therefore, it is essential to somehow memorize T11, T12, T21 and T22 so that it is possible to get back to whichever state was active. As already mentioned previously, in order to deal with history feature to influence only certain levels of the hierarchy, one can use appropriate number of H symbols applied to the desired level.

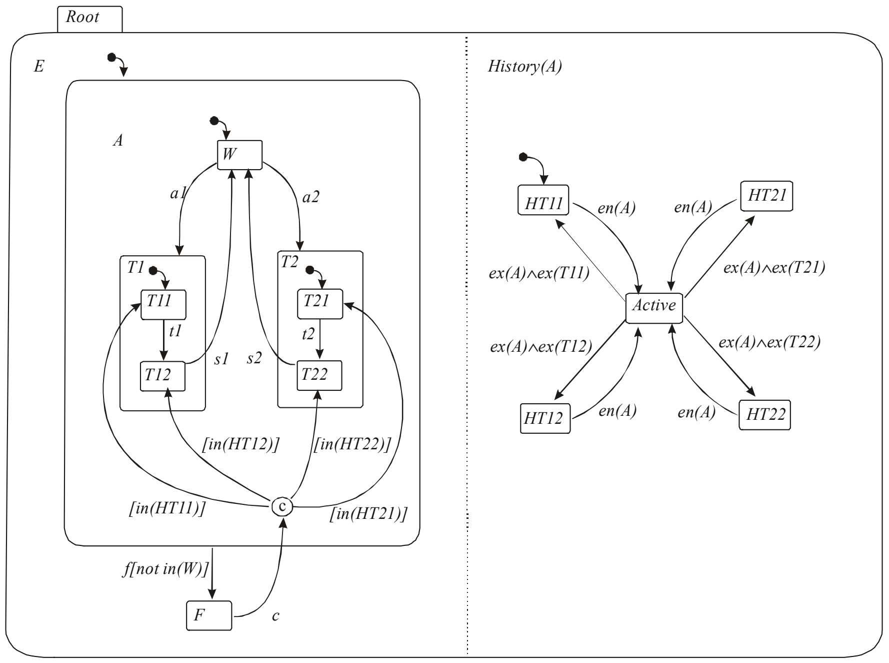

Figure 8. Dealing with H* of Figure 1b

5. Example of a Manufacturing System with History feature

Consider a flexible manufacturing system with two machines M_a and M_b operating in series and continuously producing a single product, a robot Rb and an operator Op. Each product goes through a process by the first machine M_a and is followed by another process by the second machine M_b. The products are loaded and unloaded by the robot Rb. The machines and the robot are subject to failure and the operator Op is used to repair them. The involved times in the process, i.e., the time to failure of the machines or the robot, the corresponding times to repair the failures and the times to process each product are considered to be exponentially distributed.

The machines M_a and M_b are modeled by the states W (Waiting), P (Processing), WU (Waiting to be unloaded) and B (Failure). The initial state of each machine is W. Event c_a (respectively c_b) loads a product to be processed by machine M_a (respectively M_b) and triggers a transition from W to the P state. Both the machines can operate at the same time but on different products. Machines leave their P state through two events: either a product has been processed (β_a e β_b) or a failure (λ_a e λ_b) has occurred. It is assumed that the machines are subject to failure only while in operation. Once a product is processed, the machines are switched to WU state. If a failure occurs, the machine is switched from P state to B and in this case, a product in process can be lost with probability p_a for M_a (with p_b for M_b). In state WU, the machines have to wait for the robot to unload the processed product and when this happens, event d_a or d_b (machine M_a or M_b, respectively) is triggered and a transition from state WU to W takes place. In the failure state B the operator repairs the machines and triggers events r_a or r_b (M_a or M_b, respectively) to indicate termination of repair. Then, a switch to state W takes place if the product being processed is lost or a switch to state P takes place if the product being processed is not lost (i.e. the product continues to be processed).

Robot's states are W (Waiting), La (Loading M_a), Ua (Unloading M_a), Lb (Loading M_b), Ub (Unloading Mb), and B (Failure) and the initial state is W. Robot has a priority to unload M_b (δ_b) followed by unloading M_a (δ_a) and loading M_b (γ_b) and finally loading M_a (γ_a). The robot may fail (λ_r) while in La, Ua, Lb, Ub in which case it moves to B state waiting to be repaired. Once it is repaired, it has to bypass the initial state and return to the last activity. This feature is depicted by the H symbol in Figure 9.

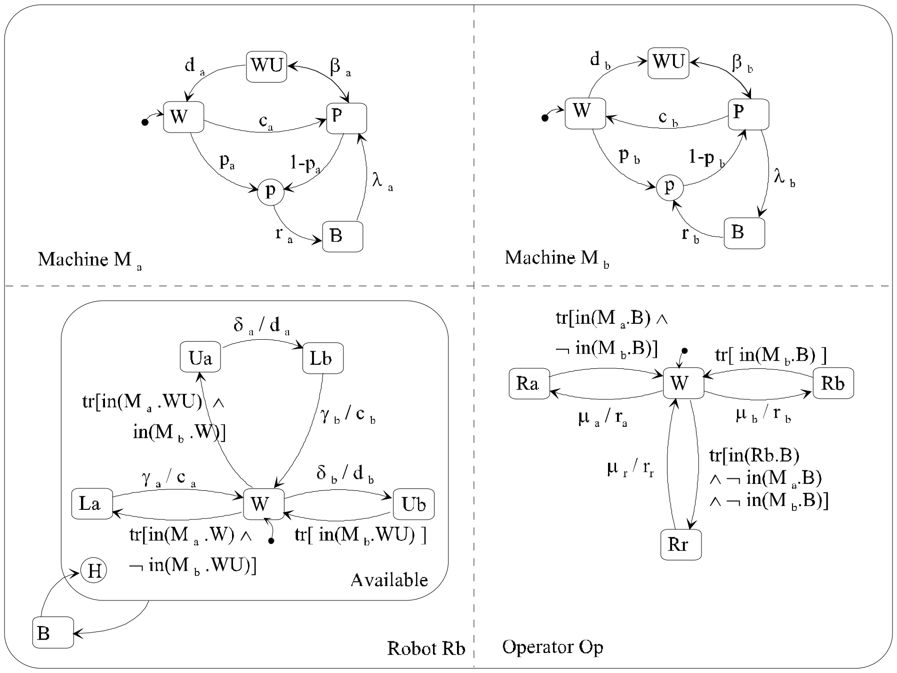

Figure 9. Statecharts representation of a Flexible Manufacturing System

The states for the operator Op are W (Waiting), Ra (Repairing M_a), Rb (Repairing M_b) and Rr (Repairing robot). A highest priority is given to repair M_b followed by a lower priority to repair M_a and the lowest priority is to repair the robot. Once in the W state, the operator reacts immediately to any failure in the system and switches to the corresponding repairing state. The times to repair are exponentially distributed with parameters μ_a, μ_b, and μ_r for M_a, M_b and for the robot respectively. Once the repair is over, the operator generates an event (r_a for the machine M_a, r_b for the machine M_b and r_r for the robot) and returns to the initial state W.

Input parameters as well as performance measurements obtained by using the software developed to generate steady-state probabilities for systems specified in Statecharts are shown respectively in Table I and Table II.

	M_a	M_b	Robot
Production rate	$\beta_a = 50$	$\beta_b = 10$	
Failure rate	$\lambda_a = 1$	$\lambda_b = 1$	$\lambda_r = 0.5$
Repair rate	$\mu_a = 10$	$\mu_b = 10$	$\mu_r = 10$
Prob. of losing a product	$p_a = 0.05$	$p_b = 0.02$	
Loading Ma			$\gamma_a = 100$
Unloading Ma			$\delta_a = 100$
Loading Mb			$\gamma_b = 100$
Unloading Mb			$\delta_b = 100$

Table I – Input values of the Model of Figure 9

	M_a	M_b	Robot
Average production rate	6.803	6.789	
Average rate for product loss	0.007	0.014	
Availability	98.6%	93.1%	98.6%

Table II – Performance Measurements

6. Conclusions

Statecharts have potential features to represent performance models due to their visual appeal. As already mentioned a software is being used in order to obtain performance measurements for systems specified in Statecharts. Recently the feature of "remembering the last visited state" bypassing the default state has been included to the software and this feature is very essential in enhancing the representation of performance models.

The idea of including memory to models that will be associated with Markov chains is not new. However, the paper described a solution of how to automatically bring the "past" to the present based on an elegant Statecharts representation. Naturally, as usually occurs in Markov modeling, this leads to paying for the extra computational effort and it will increase proportionally to the number of History symbols present in the model.

At the moment, a given specification has to be fed to the software via a main program which is quite uncomfortable depending on the size and the logic of the model. Work is in progress to develop a user friendly graphical interface. A proper formalism in order to come up with a stochastic extension to Statecharts is also under way.

References

Carvalho, S.V.; Noyes, D.; Sahraoui, A.E.K. & Soler, F. (1991) Statecharts representation of Markov models and its applications. **In**: Proceedings of 3rd International Symposium On System Research, Informatics, and Cybernetics, Baden-Baden, Germany.

Carvalho, S.V. (1991) Modeles Stochastiques appliques a l'optimisation de la performance et la surete de fonctionnement des systemes de production. Toulouse, França. Tese (Doutorado em Automática) – L'Universite Paul Sabatier de Toulouse

Chiola, G.; Marsan, M.; Conte, G. (1993) Generalized Stochastic Petri Nets: A definition at the net level and its implications. *IEEE Transactions on Software Engineering*, **19**(2), 89-106

Francês, C.R.L.; Vijaykumar, N.L.; Santana, M.J.; Carvalho, S.V.; Santana, R.H.C. (2000) Stochastic extension to Statecharts for representing performance models: an application to a file system. **In**: Proceedings of 12th Symposium on Computer Architecture and High Performance Computing, São Pedro, Brazil, 2000, p. 365-372

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

Harel, D. (1987b) *Algorithmics: The Spirit of Computing*. Addison-Wesley, England.

Harel, D.; Pnueli, A.; Schmidt, J. & Sherman, R. (1987) On the formal semantics of Statecharts. **In**: Proceedings of IEEE Symposium On Logic In Computer Science, Ithaca, USA.

Harel, D. & Naamad, A. (1996) The STATEMATE Semantics of Statecharts. *ACM Transactions on Software Engineering*, **5** **(**4), 293-333.

Harel, D. & Politi, M. (1998) *Modeling Reactive Systems with Statecharts: the Statemate Approach*. McGraw-Hill, USA.

Menascé & Almeida (1998) *Capacity Models for Web performance – Metrics, Models & Methods*. Prentice-Hall, Inc.

Kleinrock, L. (1976) *Queueing Systems*. v. 2, New York, USA: John Wiley & Sons.

Philippe, B.; Saad, Y. & Stewart, W.J. (1992) Numerical Methods in Markov Chain modeling. *Operations Research*. **40** (6), 1156-1179.

Silva, E.A.S. & Muntz, R.R. (1992) *Métodos Computacionais de Solução de Cadeias de Markov: Aplicações a Sistemas de Computação e Comunicação***.** UFRGS, Gramado.

Vijaykumar, N. L.; Carvalho, S.V. & Abdurahiman, V. (2002). On proposing Statecharts to specify Performance Models. *ACCEPTED FOR PUBLICATION IN International Transactions in Operational Research*. (***Submitted in 1999****)*.

Vijaykumar, N.L. (1999) *Statecharts: Their use in specifying and dealing with Performance Models*. Tese (Doutorado em Engenharia Eletrônica e Computação – Área de Informática), IEC-ITA, São José dos Campos.

Vijaykumar, N.L.; Carvalho, S.V.; Andrade, V. M. B.; Abdurahiman, V. (2000a) *Probabilistic Statecharts*, **In**: Proceedings of Abstracts of EURO XVII (European Conference on Operational Research), Budapest, Hungary.

Vijaykumar, N. L.; Carvalho, S.V. & Abdurahiman, V. (2000b) Specifying Manufacturing Cell with Robots and obtaining performance measurements: an approach in Statecharts and Petri nets. **In**: Proceedings of VI International Conference on Industrial Engineering and Operations Management, São Paulo, 349-355.

Vijaykumar, N. L.; Carvalho, S.V. & Abdurahiman, V. (2001) – Towards obtaining performance measurements of a manufacturing system specified in Statecharts. *Pesquisa Operacional*, ***(Submitted)***.