Gest. Prod., São Carlos, v. 19, n. 4, p. 663-675, 2012

Memory embedded in Markov models specified in Statecharts: simulation versus analytical approaches

Tratamento de memória em modelos Markovianos especificados em Statecharts: abordagens por simulação e analítica

Nandamudi Lankalapalli Vijaykumar[1]
Gian Ricardo Berkenbrock[2]
Solon Venâncio de Carvalho[1]
Valéria Maria Barros de Andrade[3]
Gurrala Veereswara Swamy[4]
Mokka Jagannadha Rao[5]

Abstract: Statecharts are a graphical representation to model reactive systems that respond to external or internal stimuli by changing the state of a given system. Statecharts can be seen as an extension of state-transition diagrams that allows modeling hierarchy, orthogonality, and interdependence. Due to their features to represent reactive systems, Statecharts have been adapted to represent and deal analytically with performance models (reactive systems whose performance is to be evaluated). An interesting feature present in Statecharts is to record the system's state, which cannot be represented in Markov models in a straightforward manner due to its "memory-less" property. The contributions of this paper are: show that Statecharts are a feasible alternative to specify a reactive system so that its performance can be evaluated by both analytical and simulation techniques; show that the inclusion of the memory representation in the Statecharts specification can indeed be made by both analytical and simulation techniques. The results of a case study of a manufacturing system show that the objectives are achieved.

Keywords: Performance models. Statecharts specification. Memory representation. Analytical solutions. Continuous-time Markov chains. Simulation.

Resumo: *Statecharts representam graficamente sistemas reativos que respondem aos estímulos externos ou internos e mudam estados de um dado sistema. Statecharts estendem diagramas de estado com hierarquia, paralelismo e interdependência. Devido às suas características, eles foram adaptados para representar e tratar analiticamente modelos de desempenho (sistemas reativos cujo desempenho deve ser avaliado). Uma característica presente em Statecharts é registrar (ou memorizar) um estado do sistema que não é possível representar numa forma direta em modelos Markovianos devido à sua propriedade de "sem memória". São duas as contribuições deste artigo: mostrar que Statecharts são viáveis para especificar sistemas reativos e avaliar o seu desempenho tanto por técnicas analíticas quanto por simulação; mostrar que a inclusão de representação de memória em Statecharts pode, de fato, ser tratada por abordagens analíticas e de simulação. Um estudo de caso de um sistema de manufatura é considerado para mostrar que os objetivos foram alcançados.*

Palavras-chave: *Modelos de desempenho. Statecharts. Representação de memória. Soluções analíticas. Cadeias de Markov a tempo contínuo. Simulação.*

[1] Laboratory of Computing and Applied Mathematics, National Institute for Space Research, CP 515, CEP 12245-970, São José dos Campos, SP, Brazil, e-mail: vijay@lac.inpe.br

[2] Department of Computer Science, Santa Catarina State University – UDESC, CEP 89219-710, Joinville, SC, Brazil, e-mail: gian@joinville.udesc.br

[3] Mentor Tecnologia, Av. Adhemar de Barros, 633, CEP 12245-010, São José dos Campos, SP, Brazil, e-mail: valeria.andrade@mentortec.com.br

[4] Department of Computer Science, Githam University, Visakhapatnam, AP, India, e-mail: drgv_swamy@yahoo.co.in

[5] Delta Studies Institute, Andhra University, Visakhapatnam, AP, India, e-mail: mjrao_isa@yahoo.co.in

Recebido em 30/11/2011 — Aceito em 17/9/2012

Suporte financeiro: PROCAD/CAPES (Projeto NF 776/2010).

1 Introduction

In systems design, usually, analysts or engineers must be capable of predicting the system performance, so that it is possible to avoid possible bottlenecks, as well as understanding the system behavior, both in the best and the worst case scenarios. An option to deal with this issue is to depend on the knowledge and expertise of the system administrator. An alternative is to use performance models so that one can infer the behavior of the system. Performance models are those models which are reactive systems from which their behavior can be determined. Such models can be applied for capacity planning, manufacturing systems performance analysis, alternative systems, capacity expansion, spare parts inventory management (GOMES; WANKE, 2008) and maintenance systems (DE MARCHI; CARVALHO; MORAIS, 2001). These application areas play a key role in production management since they enable engineers to conduct studies on different options by changing the performance models.

Several performance evaluation methods are available in the literature, and they are generally associated to stochastic processes that are mathematically well defined and can be handled computationally. Such methods use approaches such as measuring or modeling. Measurements, benchmarking, and prototyping are measuring approaches. Modeling uses a specification method to represent a given system behavior, and then a technique – either by simulation or by analytical approach – is associated to the specification; one issue is the representation of performance models.

Usually systems whose performance has to be evaluated fall into the category of reactive systems. Reactive systems are those that respond to external or internal stimuli also known as events. Events are a perturbation in the behavior, and they change the present state (for instance, *idle* to *processing*). One way to represent reactive systems is a Finite State Machine (FSM) whose representation is achieved through a state-transition diagram. FSM consists of states and transition between states. The change of one state to another occurs based on a label on the transition which corresponds to an event. Therefore, if a system whose performance is to be measured can be represented by an FSM, then it is possible to associate this representation with a technique (analytical approach or simulation) in order to obtain its performance evaluation. In case of a technique by means of analytical approach, Markov chains are usually employed. FSM resembles very much a Markov chain since a Markov chain is also represented as a set of states and transitions driven by events (continuous-time Markov chain) or probabilities (discrete-time Markov chain).

Modern complex systems require explicit representation of depth and parallelism. A clear visualization of systems with several parallel components becomes difficult by means of state-transition diagram thus requiring a consideration of using higher-level techniques (queuing networks (KLEINROCK, 1976), Generalized Stochastic Petri Nets (CHIOLA; MARSAN; CONTE, 1993) and Statecharts (HAREL, 1987).

Our proposal uses the Statecharts approach to represent and deal with performance models (VIJAYKUMAR; CARVALHO; ABDURAHIMAN, 2002). Performance models have been used to observe the behavior of a reactive system to understand these systems without the need of physically implementing them Accordingly, some metrics can be very interesting for determining performance, for instance, response time, throughput, utilization, etc.

Now, the problem is not only to represent the model, but also to solve it so that performance metrics of interest can be obtained. With respect to this, the PerformCharts tool was used to analytically determine these metrics. However, in performance models, it is common to face a situation where after a component is turned off, a return to "the last active state" of that component is necessary. As an example, let's consider a model in which a robot is responsible to do some service. In the event of a breakdown, the robot has to be repaired, and it should return to the state where it was before the breakdown to finish the interrupted task. This is easily represented in Statecharts, as shown in the following sections. The problem is to solve the model using Markov chain. However, the memory-less property of Markov chains – future depends only on the present and not on the past – has to be handled. Markov chains are simple and can deal with uncertainty enabling analyzing complex systems in the areas of production, industrial engineering mathematics, computer science, operations research, and others (SHESKIN, 2010; HERMANNS et al., 2000). In the area of production control, the task of dealing with varying service times to properly estimate manufacturing times is not an easy task. Therefore, Markov chains in queuing theory added to analysis of real queue systems with layout constraints have generated a hybrid model to access such systems (ABEDI et al., 2010).

Thus, this study aims at discussing how memory can be automatically eliminated from Statecharts models when converting the specification (from Statecharts models) in a Markov chain (VIJAYKUMAR et al., 2002). Vijaykumar, Carvalho and Abdurahiman (2002) proposed the use of Statecharts to represent Markov chains without dealing with the memory issue, which is a very interesting feature of Statecharts. Only one algorithm was discussed in Vijaykumar et al. (2002). However, a real implementation was not properly done using adequate testing in several reactive systems . In addition, the same specification is used in Statecharts to generate the performance evaluation by adopting the simulation approach. The PerformCharts tool deals only with analytical solution. It is necessary to incorporate the solution via Simulation as well. The advantages of this approach are to study the feasibility of incorporating this into the tool and to validate the PerformCharts tool. In order to deal with the simulation solution, the system simulation model was developed, and the simulation executive is based on the three-phase approach specified by Pidd (1998). This modeling approach maps two types of activity: unconditional and conditional activity. Unconditional activity also known as B activity is processed at the due simulation time, and conditional activity or C activity is processed only when the condition is accepted. There is also another activity, A activity or A phase, which is concerned with time advancement or time scan.

The next section briefly introduces Statecharts, in particular, how the memory feature is represented. In order to adapt Statecharts to be used in performance evaluation, a tool named PerformCharts has been developed. Section 3 shows the tool by describing how the Statecharts representation is handled to obtain performance measures. Section 4 deals with the memory feature represented in Statecharts and how this has been implemented in the PerformCharts tool. A case study as well as its solution to obtain performance metrics through PerformCharts is presented in Section 5. As previously mentioned, the PerformCharts tool deals with performance evaluation using only an analytical approach, and it is expected to include the solution by means of Simulation. Therefore, a simulation solution is developed in order to compare the results from an analytical approach without implementation of PerformCharts. The simulation approach has also been based on the Statecharts representation. This is presented in Section 6. Finally, some conclusions about the use of a high-level specification technique and its association with both Markov chain and simulation approach are presented.

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. Moreover, Statecharts are formal (HAREL, 1987) and (HAREL; POLITI, 1998), and their syntax and semantics enable considering complex logic to represent the behavior of reactive systems. They extend state-transition diagrams by incorporating notions of hierarchy (depth), orthogonality (represent-tation of parallel activities), and interdependence (broadcast-communication).

The states are clustered to represent depth. This feature enables combining a set of states with common transitions into a macro-state also known as super-state. Super-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 *AND* decomposition represents concurrency, i.e., when a super-state is active, all of its sub-states are active. A state is *BASIC* when there are no further refinements from it.

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. A complete syntax and semantics of Statecharts are described in Harel (1987), Harel et al. (1987) and Harel and Politi (1998).

By definition, when modeling a given system, there must always be an initial state, a default state, in Statecharts. Thus, the initial configuration consists of active basic states of each orthogonal component. However, the default state can be bypassed by using *history*, i.e. when a system becomes active the most recently visited state becomes active. This is indicated by the symbol H. It is also possible to use the history all the way down to the lowest level as defined in the Statecharts formalism (HAREL, 1987). In this case, the symbol H* is used. In order to influence only certain levels of abstraction, the symbol H should be applied to the appropriate level. The history feature is shown in Figure 1.

Let's consider equipment E that has as a sub-state a macro-state A, which is an XOR state. Its sub-states are W (idle), and the macro-states T1, and T2 that represent two types of jobs T1 and T2 have further refinements into basic states T11 and T12 and T21 and T22, respectively. E behaves as follows: when idle (state W), it can be requested to process either T1 or T2. Event *a1* triggers the job type T1, whereas event *a2* triggers type T2. End of service is represented by events *s1* and *s2* for jobs T1 and T2, respectively. Both types of jobs have to undergo two sub-processes, T11 and T12 (in case of T1) and T21 and T22 (in case of T2). When the equipment is busy, it may fail, and this takes to state F.

(Failure) via event *f*. Note that the guard condition *not in*(*W*) is attached to guarantee that the equipment may fail only when it is not idle.

When the failure is corrected, the last state visited becomes active, represented by H. In Figure 1a, when the super-state A becomes active, the most recently visited state (T1 or T2) will become active bypassing the default state. When T1 is the most recently visited state, the sub-state T11 will become active since it is the default state within T1. With regard to T2, sub-state T21 (default state within T2) is the one that will become active. In Figure 1b, H* is used to indicate that all levels within the hierarchy will be affected. For example, if T2 is the most recently visited state, either sub-state T21 or T22 will become active.

The literature shows several applications of Statecharts. They have become quite popular due to their visual appeal making them feasible to describe complex logic in reactive systems. Recently, Li, Tong and Nian (2010) have formalized Statecharts semantics including history states employing temporal description logic to be used in validation. Haschemi (2009) proposed coverage criteria for black box testing for model-based testing for Statecharts. History is taken into consideration in these coverage criteria. Wagstaff, Peters and Scharenbroich (2008) proposed a system to convert text-based specification into UML Statecharts in order to generate code in C or C++. They consider entry by history specification as well. Vijaykumar et al. (2006) proposed a solution to specify history states in Statecharts for Performance Evaluation based on Markov chains. However, they have never implemented or validated this approach.

3 PerformCharts: construction of a continuous-time Markov chain from a Statetcharts model

A continuous-Time Markov Chain consisting of transition rates among states is solved through numerical methods (SILVA; MUNTZ, 1992; PHILIPPE; SAAD; STEWART, 1992) to determine the steady-state probabilities. Thus, the problem is solved if the model represented by Statecharts is converted into a Markov chain that corresponds to the behavior of the specified model (VIJAYKUMAR; CARVALHO; ABDURAHIMAN, 2002). Therefore, the PerformCharts tool is responsible to automatically generate a Markov chain from a Statecharts specification.

Once a model is represented by Statecharts, the enabled events must be stimulated so that new configurations (set of active states of all parallel components) are obtained. Internal events (*true*(*condition*), *false*(*condition*) – *abbreviated respectively as tr and fs* – and actions) are automatically sensed and stimulated. External events must be explicitly stimulated to describe the dynamics of the modeled system behavior. In order to enable the association of Statecharts model with a Markov chain, the only type of external events that can be considered are stochastic events. The time between their activation and their occurrence follows a stochastic distribution, which, in particular, has to be exponential.

An example is used to illustrate the process of converting Statecharts specification into a Markov chain. Let's consider a system with three parallel components that correspond to two machinery equipment and a supervisor to repair any failure in the equipment, as shown in Figure 2.

Components E1 and E2 denote the equipment, whereas the component Supervisor repairs the equipment when it fails and a priority is provided to repair E1 whenever there is the occurrence of double equipment failure. E1 and E2 are XOR macro-states and have similar behavior to that of the basic states W1, P1, and B1 and W2, P2, and B2, respectively. W1 and

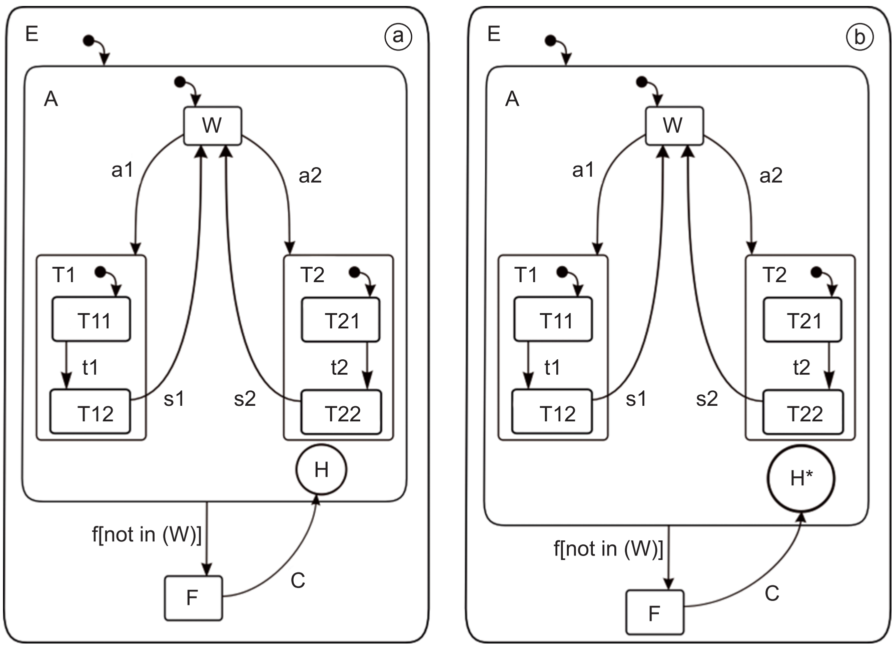

Figure 1. a) History in a single level; b) History in all the hierarchy levels.

W2 represent that E1 and E2 are idle, and processing a job means that these components, equipment E1 and equipment E2, are in P1 (and P2). Both E1 and E2 are subject to failure, and in this case, the states P1 and P2 move to states B1 and B2. Supervisor is an XOR macro-state, and its idle state is represented by WS. C1 and C2 represent, respectively, repairing E1 and E2. The list of stochastic events (exponentially distributed) includes a1, r1, f1, s1 a2, r2, f2, and s2. Internal events are tr[in(B1)] and 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 (without delay time, *i.e.*, immediate transition). A brief explanation of the events is given in Table 1.

Therefore, 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 example in Figure 2, there are no active internal events for the initial configuration (W1, W2, WS). Hence, the external events have to be listed. For the initial configuration, the enabled stochastic events are a1 and a2. Therefore, these events will be stimulated or triggered so that transitions are fired to yield new configurations.

When the system is in the initial configuration and a1 is triggered, the next configuration is P1, W2, and WS. Suppose that during the process of stimulating the events, the resulting configuration is (B1, B2, WS). In this case, the active events are the so called immediate events tr[in(B1)] and tr[(in(B2) ^ ¬in(B1)]. 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 those that have to be stimulated.

The corresponding state-transition diagram is shown in Figure 3. As one can notice, this diagram contains only stochastic events that follow exponential

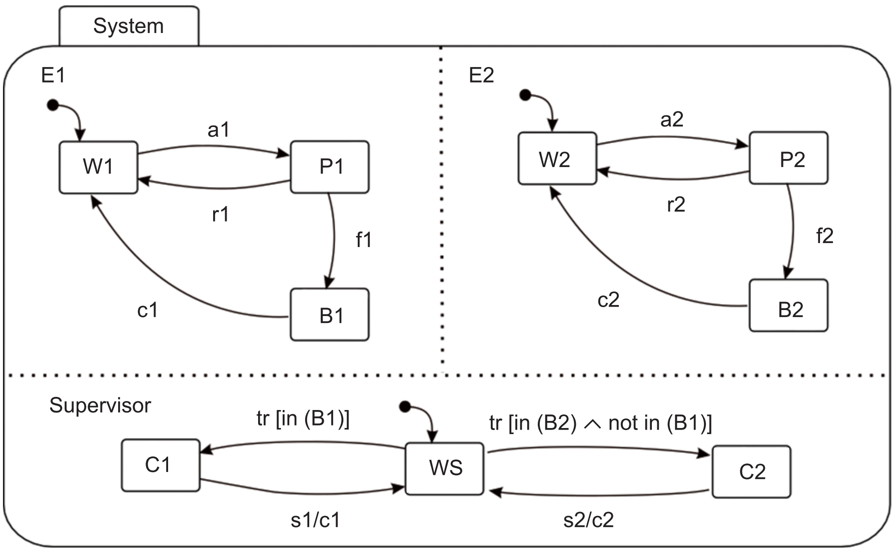

Figure 2. Statecharts representation of an equipment with a repairer.

Table 1. Brief explanation of events in Figure 2.

Event	Type	Meaning
a1	Stochastic	Arrival of a job for E1
r1	Stochastic	End of service by E1
f1	Stochastic	Failure of E1
c1	Internal	E1 repaired
a2	Stochastic	Arrival of a job for E2
r2	Stochastic	End of service by E2
f2	Stochastic	Failure of E2
c2	Internal	E2 repaired
tr[in(B1)]	Internal	B1 (in E1) is active
tr[in(B2) ^ not in(B1)]	Internal	B2 (in E2) is active AND B1(in E1) is NOT active
s1	External	End of repair of E1
s2	External	End of repair of E2

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 metrics.

According to the example previously shown, it should be clear that the translation from a Statecharts representation into a Markov chain is not a "blind" AND product. Otherwise, the Statecharts in Figure 2 would result into a 27-state diagram instead of 10-state diagram (Figure 3). However, one must consider the well-known problem of state-space explosion that still exists and depends very much on the number of components and number of sub-states within each component. The advantage is that the Statecharts specification of the model is represented in a "cleaner" fashion instead of a web of arcs traversing among several hundreds of states.

Questions may arise with respect to the process of reaction. Especially, what happens when more than one stochastic event in separate components become active. These events are put into a sequence and stimulated one after another. So far, all examples tested generated the same Markov chain regardless of the order in which the events are stimulated. This can be seen in Figure 3. By taking any configuration, at which more than one exponentially distributed stochastic event exists, the resulting configurations are the same regardless of the order in which the events are stimulated. However, in order to guarantee this, a formal proof has to be considered and developed which is not within the scope of the present study. There might be cases in which the logical behavior of a complex system depends on the order of the events; then a time parameter (associated to an event) has to be defined or some type of alternative ways of synchronization must be made.

The subset implemented in PerformCharts is shown in Table 2.

4 PerformCharts: handling memory in Markov models represented in Statecharts

One powerful feature provided in Statecharts is the Entry by History. Due to the memoryless property in the Markov chains, Entry by History seems, at a first glance, not to be compatible with Markov models since it has to depend on the past. However, 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, the representation of performance models can be very much improved (VIJAYKUMAR et al., 2002). 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, let's consider the example in Figure 4 in which the history feature is depicted.

Figure 4 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 to service type T2, and event a3 to service type T3. The equipment may fail, but it is assumed that the failure occurs during the processing, and the 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). Figure 4 shows the representation of an equipment that when returning from F state to P

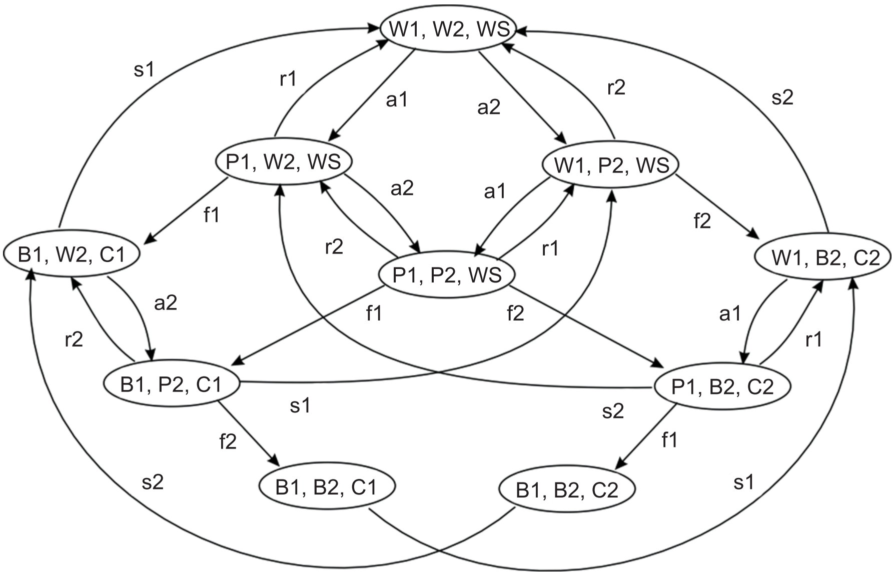

Figure 3. Markov chain of Figure 2.

Table 2. Subset of Statecharts supported by PerformCharts tool (SANTIAGO JÚNIOR, 2011).

Syntax feature	Support
States	X
Shallow and deep History	X
Hierarchy function	X
Type function	X
Default function	X
Expressions: If k is a number then $k \in V$	X
Expressions: If $v \in V_p$ then $v \in V$	X
Expressions: If $v \in V$ then $current\ (v) \in V$	
Expressions: If $v_1,v_2 \in V$ and op is an algebraic operation then $op(v_1,v_2) \in V$	X
Conditions: $T, F \in C$, T, F stand for *true*, *false*, respectively	X
Conditions: If $c \in C_p$ then $c \in C$	X
Conditions: If $s \in S$ then $in(s) \in C$	X
Conditions: If $e \in E$ then $not\_yet(e) \in C$	
Conditions: If $u, v \in V$, $R \in \{=, >, <, \neq, \leq, \geq\}$ then $u\ R\ v \in C$	X
Conditions: If $c \in C$ then $current\ (c) \in C$	
Conditions: If $c_1, c_2 \in C$ then $c_1 \vee c_2, c_1 \wedge c_2,\ \neg c_1 \in C$	X
Events: $\lambda \in E$, λ is the *null* event	X
Events: If $e \in E_p$ then $e \in E$	X
Events: If $c \in C$ then $true(c) \in E$	X
Events: If $c \in C$ then $false(c) \in E$	
Events: If $v \in V$ then $changed(v) \in E$	
Events: If $s \in S$ then $exit(s)$, $entered(s) \in E$	X
Events: If $e_1, e_2 \in E$ then $e_1 \vee e_2, e_1 \wedge e_2 \in E$	X
Events: If $e \in E$, $c \in C$ then $e[c] \in E$	X
Actions: $\mu \in A$, μ is the *null* action	X
Actions: If $c \in C_p$, $d \in C$ then $c: = d \in A$	
Actions: If $v \in V_p$, $u \in V$ then $v: = u \in A$	X
Actions: If $a_i \in A$, $i = 0, ..., n$ then $a_0; ...; a_n \in A$	
Labels	X
Transitions	X

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 handle this situation, the solution to be proposed has somehow to "memorize" the last active state T1, T2, or T3 before a failure occurs so that this active state can be reached after recovering from the failure (F state). The proposed solution is to create another dummy component, History(P), consisting of the states that are essential to history plus one more Active state. This extra state indicates that 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 the occurrence of a failure is "memorized" by making the corresponding state in the dummy component active.

One can notice that in the dummy component, the solution used events already defined in the Statecharts formalism such as *entered*(*X*) and *exit*(*X*) *– abbreviated respectively as en*(*X*) *and ex*(*X*) *where X is a State* – which respectively means that they are stimulated each time state X is entered and when state X is exited, as shown in Figure 5. In the event of P macro-state becomes inactive 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. Regardless of the destination state from P (either F or W), T1 is "memorized". The "memory" aspect will be used only when returning from the F state.

Note that in the proposed solution, a dummy state, History(E), must be created for each state E that contains Entry by History. In general, 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. The Markov chain of this example is shown in Figure 6.

There is also H*, which refers to Entry by History affecting all the levels within a hierarchy. In order to illustrate this aspect, the example shown in Figure 1 is considered. Figure 7 shows the addition of another orthogonal component in order to deal with H of Figure 1a. Just the macro-states T1 and T2 are necessary to be remembered due to the use of the symbol H in Figure 1a. Once T1 or T2 are active, their default states (T11 or T21) will be active anyway. Therefore, "memorizing" T1 or T2 is enough.

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 "memorize" 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 previously, mentioned in order to deal with the history feature to influence only certain levels of the hierarchy, one can use an appropriate number of H symbols applied to the desired level. Algorithms to deal with history represented in Statecharts and its consequent translation into a Markov chain can be seen in Vijaykumar et al. (2006).

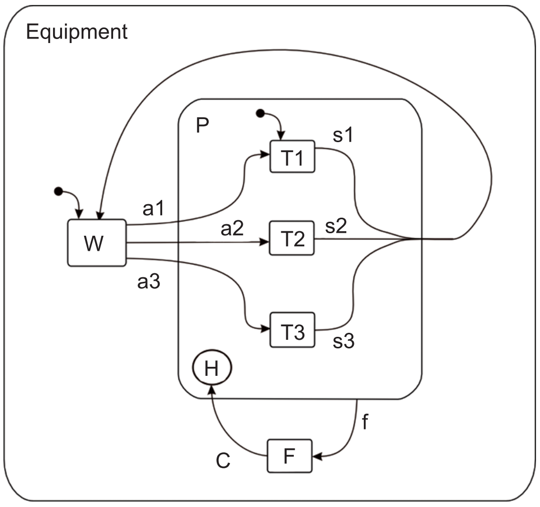

Figure 4. Entry by History feature in a model.

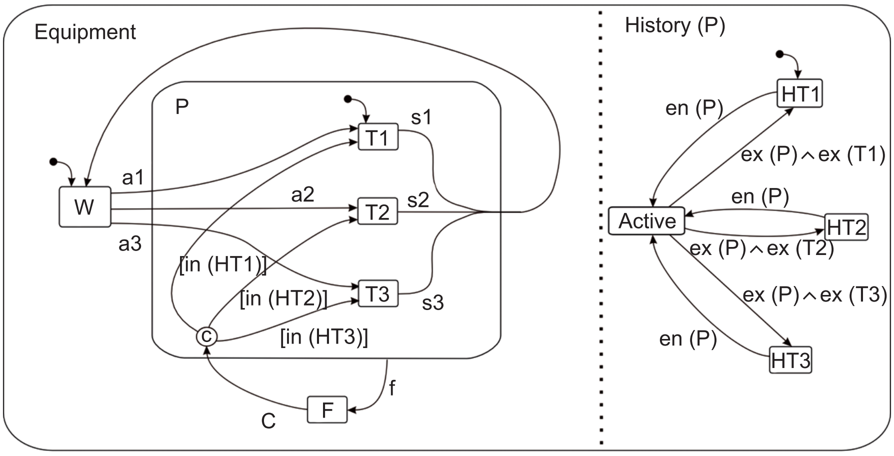

Figure 5. A solution to Entry by History.

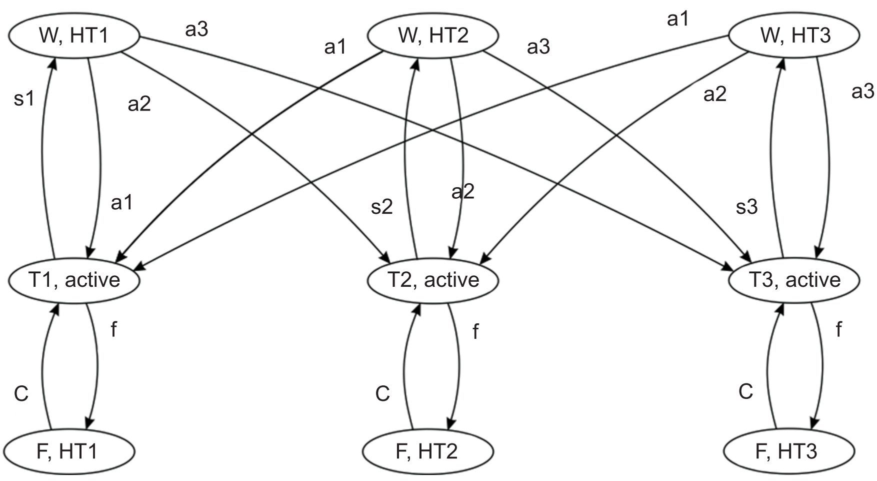

Figure 6. Markov chain of Figure 5.

5 Case study: PerformCharts to evaluate a performance evaluation of a manufacturing system with memory

Let's 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 times in the process are: time to failure of the machines or the robot, the corresponding times to repair the failures, and the times to process each product. All of them are considered to be exponentially distributed.

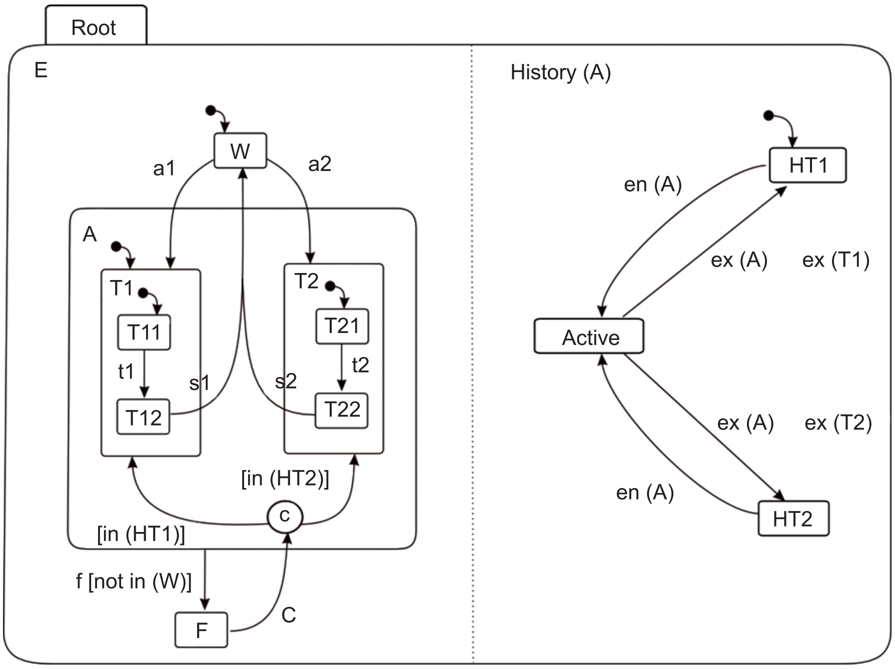

Figure 7. Dealing with History of Figure 1a.

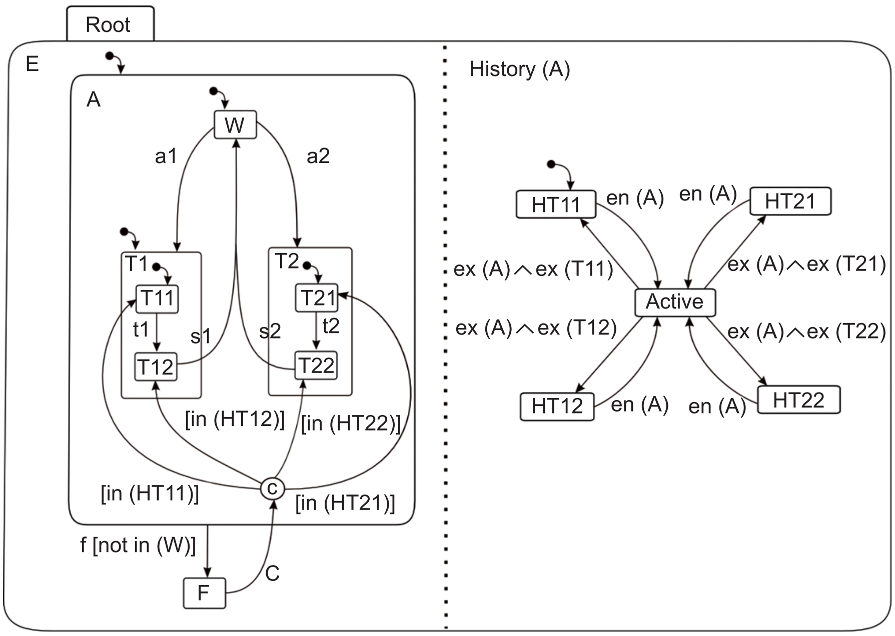

Figure 8. Dealing with H* of Figure 1b.

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 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 and β_b) or a failure (λ_a and λ_b) has occurred. It is assumed that the machines are subject to failure only while they are 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 M_b), and B (Failure), and the initial state is W. The 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 Ma ($\bullet\gamma_a$). 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.

The states for the operator Op are W (Waiting), Ra (Repairing M_a), Rb (Repairing M_b), and Rr (Repairing robot). The 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, for 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 and the performance measurements obtained using the software developed to generate steady-state probabilities for systems specified in Statecharts are shown respectively in Tables 3 and 4.

6 Performance evaluation from simulation

The manufacturing system presented previously was considered for the simulation study, and then the required computer programming has been

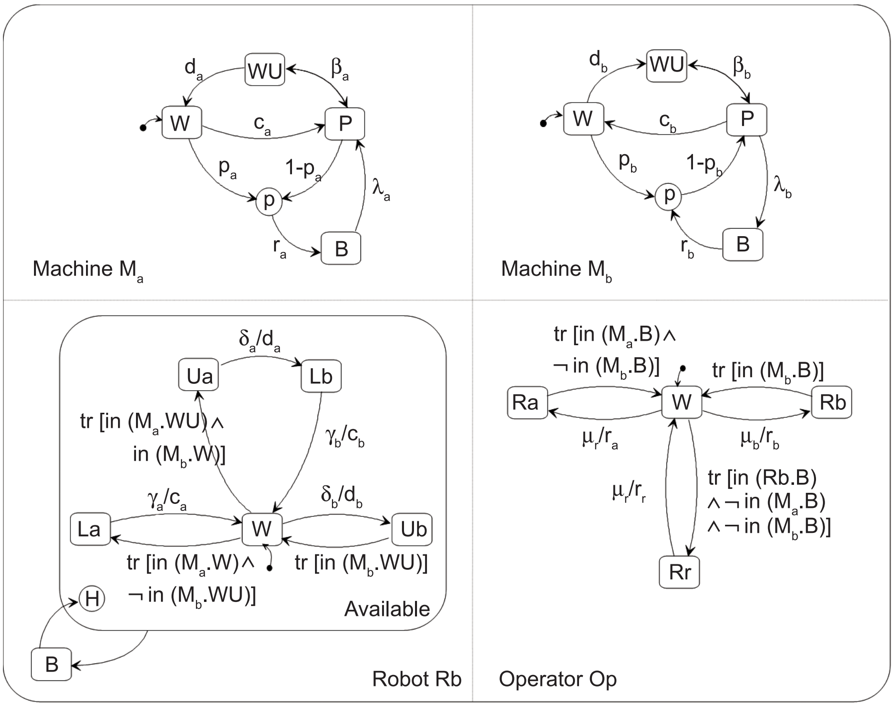

Figure 9. Statecharts representation of a Flexible Manufacturing System.

developed to perform this simulation. The approach described by Pidd (1998) was followed for the simulation solution.

Additionally, the system in Figure 9 was modeled using the three-phase approach (TOCHER, 1963). Tocher shows the simulation executive as a process with three phases: time scan (A-activity – it manages the time advancement), execution of B-activities (unconditional – they execute their related procedure) that are due at the current time, and execution of C-activities (conditional – they only execute if their processing condition are satisfied). To model using this approach, it is necessary to define the entities, the resources, and the entities' life cycle. Therefore, the model is a set of entities, resources, and B and C activities that define the life cycle. Using this approach, the model has twenty B-activities and nine C-activities. The memory issue in simulation is dealt, normally, through modeling using B and C activities, as well as using attribute values to decide the state in which the product was. The A activity or time advancement is a task related to the simulation executive and it is not a model concern.

The values for the input parameters and the product arrival rate are shown in Table 3. Ten replications of the simulation were executed, and each execution lasted ten thousand units of time. Each simulation had a different product arrival rate. Hence, eighty simulations were run to obtain the results to be compared with the values shown in Table 4.

Performance measures obtained from simulation with arrival rate of seven units are shown in Table 5. The results obtained were close to the measurements shown in Table 3. Additionally, the queue size of M_b was monitored, as shown in Figure 10. This is one major advantage over the analytical approach because one can, without too much effort, monitor the system behavior. After analyzing the results, it could be noted that, based on these input parameters, the queue size of M_b begins with an unstable behavior if product arrival rate is greater than seven units.

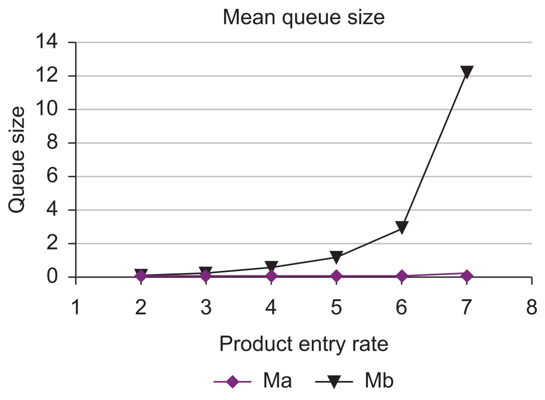

Figure 10. Queue size variations of machine B.

Table 3. Input values of the Model of Figure 9.

	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 4. Performance measurements – Markov approach.

	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 5. Performance Measurements – Simulation approach: with 7 Products arrival rate.

	Ma		Mb		Robot	
	Mean	StdDev	Mean	StdDev	Mean	StdDev
Average product rate	6.999	0.024	6.984	0.024	-	-
Average rate for product loss	0.007	0.001	0.014	0.001	-	-
Availability (%)	98.537	0.051	92.935	0.153	98.547	0.052

7 Conclusions

Statecharts have potential features to represent performance models due to their visual appeal. Considering that these models can be used for capacity planning, capacity expansion, inventory management, and maintenance system, engineers and analysts can use the high-level abstraction provided by the Statecharts to create performance models in their areas of interest. The software PerformCharts has been used in order to obtain performance measurements for systems specified in Statecharts. The feature of "remembering the last visited state" bypassing the default state has been included to the software, and this feature is essential to improve the representation and deal with performance models.

The present study described a solution to automatically bring the "past" to the present based on an elegant Statecharts representation. When specifying the models (when they are simple enough) directly as Markov chains, usually the same solution of bringing the "past" to the present is applied. The solution (just as it occurs when systems are modeled directly as Markov chain) proposed in this paper, leads to a computational effort that increases proportionally to the number of History symbols present in the model. This study also shows how to deal with this issue by adopting the simulation approach. The simulation approach is also based on Statecharts representation. The representation of memory within the simulation approach was much easier since the information about the memory is directly embedded into the model specification and implementation.

The analytical solution using Markov chains is a much faster approach as long as the exponential distribution of stochastic information (events on transition arcs) is guaranteed. However, this may not always be true in real world cases. Therefore, simulation is an alternative since there are no restrictions with respect to stochastically distributed times. Nevertheless, the drawback is the computational effort since simulation requires lot of preparation such as data collection, sampling, and several runs and statistical measurements.

Currently, PerformCharts deals only with analytical approach. It is expected to incorporate simulation solution to the tool. The tool has a textual interface based on eXtensible Markup Language (XML), and this language is converted into C++ main program. This has brought a reasonable advantage of specifying a complex system since most of the specification is based on pointers. A graphical interface is in progress, and due to one-to-one correspondence of a Markov chain to a Finite State Machine (FSM), this tool has also been used to generate test case sequences.

References

ABEDI, S. et al. Using Markov chain and Simulation to Analysis and Optimization Production Lines Systems with Layout Constraints. In: INTERNATIONAL CONFERENCE ON COMPUTERS AND INDUSTRIAL ENGINEERING - CIE, 40., 2010, Awaji. **Proceedings**... Awaji, 2010. p. 1-6.

CHIOLA, G.; MARSAN, M.; CONTE, G. Generalized Stochastic Petri Nets: A definition at the net level and its implications. **IEEE Transactions on Software Engineering**, v. 19, n. 2, p. 89-106, 1993. http://dx.doi.org/10.1109/32.214828

DE MARCHI, M. M.; CARVALHO, S. V.; MORAIS, P. R. Um modelo estocástico para a manutenção de um equipamento baseado na inspeção das peças produzidas. **Gestão & Produção**, v. 8, n. 2, p. 115-127, 2001. http://dx.doi.org/10.1590/S0104-530X2001000200002

GOMES, A. V. P.; WANKE, P. Modelagem da gestão de estoques de peças de reposição através de cadeias de Markov. **Gestão & Produção**, v. 15, n. 1, p. 57-72, 2008. http://dx.doi.org/10.1590/S0104-530X2008000100007

HAREL, D. Statecharts: a visual formalism for complex systems. **Science of Computer Programming**, v. 8, p. 231-274, 1987. http://dx.doi.org/10.1016/0167-6423(87)90035-9

HAREL, D. et al. On the formal semantics of Statecharts. In: IEEE SYMPOSIUM ON LOGIC IN COMPUTER SCIENCE, 1987, Ithaca. **Proceedings**... IEEE, 1987.

HAREL, D.; POLITI, M. **Modeling Reactive Systems with Statecharts**: the Statemate Approach. McGraw-Hill, 1998.

HASCHEMI, S. Model Transformations to Satisfy All-Configurations-Transitions on Statecharts. In: MODELS WORKSHOP ON MODEL-DRIVEN ENGINEERING, VERIFICATION, AND VALIDATION - MoDeVVa'09, 2009, Denver. **Proceedings**... Denver, 2009. http://dx.doi.org/10.1145/1656485.1656490

HERMANNS, H. et al. A Markov Chain Model Checker. **Lecture Notes in Computer Science**, v. 1785, p. 347-362, 2000. http://dx.doi.org/10.1007/3-540-46419-0_24

KLEINROCK, L. **Queueing Systems**. New York: John Wiley & Sons, 1976. v. 2.

LI, M.; TONG, G.; NIAN, F. The Semantics Research of StateCharts. In: INTERNATIONAL SYMPOSIUM ON INFORMATION PROCESSING - IFIP, 3., 2010, Qingdao. **Proceedings**... Qingdao, 2010.

PHILIPPE, B.; SAAD, Y.; STEWART, W. J. Numerical Methods in Markov Chain modeling. **Operations Research**, v. 40, n. 6, p. 1156-1179, 1992. http://dx.doi.org/10.1287/opre.40.6.1156

PIDD, M. **Computer Simulation in Management Science**. 4th ed. John Wiley & Sons, 1998.

SANTIAGO JÚNIOR, V. A. **SOLIMVA**: A Methodology for Generating Model-Based Test Cases from Natural Language Requirements and Detecting Incompleteness in Software Specifications. 2011. Doutorado (Computação Aplicada)-Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2011.

SHESKIN, T. J. **Markov Chains and Decision Processes for Engineers and Managers**. CRC Press, 2010.

SILVA, E. A. S.; MUNTZ, R. R. **Métodos Computacionais de Solução de Cadeias de Markov**: Aplicações a Sistemas de Computação e Comunicação. UFRGS, 1992.

TOCHER, K. D. **The Art of Simulation**. English Universities Press, 1963.

VIJAYKUMAR, N. L.; CARVALHO, S. V.; ABDURAHIMAN, V. On proposing Statecharts to specify Performance Models. **International Transactions in Operational Research**, v. 9, n. 3, p. 321-336, 2002. http://dx.doi.org/10.1111/1475-3995.00358

VIJAYKUMAR, N. L. et al. On embedding memory in Markov Models specified in Statecharts. In: WORKSHOP ON COMPUTING SYSTEM PERFORMANCE - WPERFORMANCE, 1.; BRAZILIAN COMPUTER SYMPOSIUM - SBC, 2002, Florianópolis. **Proceedings**... Florianópolis, 2002.

VIJAYKUMAR, N. L. et al. Performance Evaluation from Statecharts representation of Complex Systems: Markov Approach. In: WORKSHOP ON COMPUTING SYSTEM PERFORMANCE - WPERFORMANCE, 5.; BRAZILIAN COMPUTER SYMPOSIUM - SBC, 2006, Campo Grande. **Proceedings**... Campo Grande, 2006.

WAGSTAFF, K. L.; PETERS, K.; SCHARENBROICH, L. **From Protocol Specification to Statechart to Implementation**. Jet Propulsion Laboratory, California Institute of Technology, 2008.