Towards obtaining Performance Measures of Manufacturing Systems specified in Statecharts

Nandamudi L. Vijaykumar1
Solon Venâncio de Carvalho1
Vakulathil Abdurahiman2
Gian Berkenbrock2
Carlos R. L. Francês3

1 Instituto Nacional de Pesquisas Espaciais (INPE), SJCampos, SP, Brasil
2 Instituto Tecnológico de Aeronáutica (ITA), SJCampos, Sp, Brasil
3 Universidade Federal do Pará (UFPA), Belém, PA, Brasil

vijay@lac.inpe.br, solon@lac.inpe.br, rahiman@ita.br, gian@ita.br, rfrances@ufpa.br

June 18, 2007

Abstract
Performance information of a given system is essential for evaluating its behavior. It is important both for new systems to be built and for existing systems. However, in the specification of such systems, when evaluating their performance, ideas of depth and parallelism are to be taken into consideration. The objective of this paper is to show that Statecharts provide a wealth of details with clarity while making it possible to compute specifications to obtain performance measurements by using an analytical approach. An example is given to illustrate this capability. This research paper is to show that Statecharts can be considered as an alternative to specify and deal with performance information of reactive systems.

Resumo
Informações sobre avaliação de desempenho são essenciais tanto para sistemas a serem desenvolvidos quanto para sistemas já existentes. No entanto, a especificação de tais sistemas, onde a avaliação de desempenho é necessária, deve levar em consideração características como profundidade e paralelismo. Este artigo descreve a abordagem de se utilizar a técnica Statecharts para especificar um sistema e ao mesmo tempo é mostrado como deve ser tratada a especificação para gerar medidas de desempenho através de métodos analíticos. Alguns exemplos ilustram a viabilidade desta abordagem. O artigo descrito mostra que Statecharts podem ser considerados como uma alternativa para especificar e avaliar desempenho de sistemas reativos.

Keywords: Manufacturing Systems, Performance Evaluation, Markov Chains, Statecharts

1 Introduction
Generally speaking, specific features need to be considered for representing a system in a clear manner with the objective of evaluating the performance of the system. These features include hierarchy (systems consisting of subsystems or components and so on), parallelism or orthogonality (more than one subsystem working or functioning at the same moment), and interdependence
(synchronization and communication of subsystems). Systems with these characteristics have been called complex systems although other definitions are also available in (SILJAK, 1991).

Complex systems, according to the definition used in this paper, are reactive systems. The main characteristic of a reactive system is its whole behavior is based on reaction to events received by internal and external media. These complex systems are heavily based on controls or events. In these cases data is not handled just by the description of to and from activities. The reaction to different kinds of events takes place in complex ways. In fact, a vast majority of the systems in day to day applications are rather reactive in nature, see (HAREL, 1987).

Due to their nature of randomness, complex systems for which performance is to be evaluated are, in general, represented by stochastic models. In these models, the system behavior can be represented by a state-transition diagram. In this diagram the states of various components of the system (idle, busy, failure, etc.) are represented and the transitions among the states take place through the events that correspond, for example, to the termination of a job or to some disruption in the system such as a failure. One more interesting factor related to these state-transition diagrams is that they are considered as Markov Chains, if the events that lead to transition among states follow an exponential distribution. The reason is that if a stochastic model is represented by a state-transition diagram and if it is a Markov Chain, then it can be solved by using an analytical approach (SILVA; MUNTZ, 1992). This solution results in steady-state probabilities which are probability functions denoting the occupation of states during a certain period of time or in a long horizon. However, a question still remains to be answered: how to represent a complex system in a high-level fashion so that performance measures can be obtained?

The first natural choice would be state-transition diagrams since they have a correspondence with Markov Chains if events in these diagrams follow an exponential distribution. However, these diagrams have some drawbacks in modeling complex systems. For example, by covering all the combinations of a concurrent model, the diagram increases exponentially and consequently leads to a state blow-up phenomenon (DRUSINSKY, 1994). A detailed representation in state-transition diagrams may lead to a large number of states. Concurrency and interdependence among the system components become difficult to be handled because they are not represented directly. This difficulty inherent in using state-transition diagram representations brings up the necessity of developing high level methods for specifying complex systems. Among them queueing networks (KLEINROCK, 1976) and (BUNDAY, 1996) and Petri nets can be mentioned (PETERSON, 1981), (MURATA, 1989) and (MACIEL et al., 1996). Queueing networks were originally created to be applied to performance models. Petri nets have a very strong basis to represent a wide variety of systems and, in particular Stochastic Petri nets and Generalized Stochastic Petri nets are extensions of Petri nets to deal with performance models.

This work proposes the use of Statecharts (HAREL, 1987), (HAREL; POLITI, 1998) as another alternative to model the behavior of reactive systems and at the same time associate the specification with an analytical approach (more precisely Markov Chains) in order to obtain performance measurements. This method was originally developed to specify, simulate and control real-time systems. Indeed, they are an extension of state-transition diagrams by adding and enhancing concepts of hierarchy and orthogonality. Statecharts have a defined formalism and they are effective for a clear visualization of a given system (HAREL et al., 1987) and (HAREL; POLITI, 1998).

Statecharts are rich and have a lot of basic elements to represent a wide variety of aspects in modeling the behavior of a system. However, this work considered only a part of these elements essential to represent performance models. For instance, elements such as variables and expressions and some types of events and conditions have not been considered in the present paper. Software has been developed to specify a reactive system using Statecharts and to automatically generate steady-state probabilities with which performance measurements can be obtained.

The application of Statecharts as an alternative for solving the problem of specifying performance models, already hinted by (HAREL, 1987), is discussed in (CARVALHO et al., 1991), (CAR-
This paper is organized as follows: (2) a brief description of Statecharts is provided; (3) the process of generating performance measurements from a specification in Statecharts is discussed; (4) a few examples of manufacturing systems that are reactive by nature are shown to illustrate that the Statecharts specification method can also be considered as a potential candidate to represent and deal with performance models; (5) finally, conclusions and future research perspectives 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 come with a strong formalism (Harel et al., 1987) and (Harel; Politi, 1998) and their visual appeal along with the potential features enable considering complex logic to represent the behavior of reactive systems. They are an extension of state-transition diagrams and these diagrams are very much improved with notions of hierarchy (depth), orthogonality (representation of parallel activities) and interdependence (broadcast-communication).

States are clustered by means of representing depth. With this feature it is possible to combine a set of states with common transitions into a macro-state also known as super-state. Super-states are usually organized before being refined into further sub-states thus enabling a top down approach. State refinement can be achieved by means of XOR decomposition and AND decomposition. The former decomposition may be used whenever 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), and (Harel; 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, 1987). In this case the symbol $H^*$ is used.

For a more detailed description of the Statecharts formalism, see (Harel, 1987), (Harel et al., 1987), and (Harel; Politi, 1998).

3 Construction of a Continuous-Time Markov Chain from a Statecharts Model

A Markov Chain, 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 (Phillippe et al., 1992) and (Silva; Muntz, 1992) to determine the steady-state probabilities. Therefore, the problem of constructing a Markov model will be solved if the model represented in Statecharts generates the Markov chain that corresponds to the behavior of the specified model.

Algorithm, in this paper, is explained in an informal manner. Details of complete algorithms may be referred to in (Vijaykumar et al., 2006). Once the model is specified in Statecharts, the first step is to check which events are to be triggered for the initial configuration determined by default
states of each parallel component. Recalling the categories of events explained earlier, internal (or immediate) events are the ones that are automatically triggered by the internal logic of Statecharts. As long as these events are found to be active for the initial and resulting configurations, reactions continuously take place by changing one configuration to another until a configuration is reached from which no more internal events are triggered. The next step is to deal with the stochastic events which have to be explicitly stimulated. Therefore, based on the resulting configuration from internal events, stochastic events that can be enabled are listed and triggered so that transitions are fired yielding new configurations. In order to make the association of a Statecharts model with a Markov Chain, the only type of events considered to be stimulated are the stochastic events. This means that the time between activation and occurrence of events follows a stochastic distribution. In particular, for Continuous-Time Markov Chains, this distribution has to be exponential. Once a configuration is obtained, internal events, if enabled, are triggered, firing transitions to yield new configurations. In both the cases, actions also have to be considered if they are associated with these events in a transition. In this case a reaction occurs whenever appropriate, based on the action which is considered as an internal event. This process continues until all the configurations have been expanded. This process results into a structure that contains a source configuration, stimulated stochastic event (along with its rate), and the target configuration. This structure is, indeed, a Markov Chain (specified as a transition matrix) with which steady-state probabilities can be determined.

However, some comments on generating the Markov chain are in order. The analytical approach is based on Continuous-Time Markov Chains. Naturally, some restrictions are imposed. For instance, external events (explicitly triggered for a reaction to take place) carry stochastic information, in particular, exponential distribution. Therefore, at each instant, even though more than one stochastic event is enabled, only one external (stochastic) event can be triggered, i.e., two different external events cannot be triggered at the same time. The process applies a reaction on each stochastic event in sequence. However, a same external event can be triggered within more than one orthogonal component consequently affecting the states within the components where this external event is active. Using the same logic for the case of immediate events, only one such event can be active in each orthogonal component. The order of this sequence is indifferent as the resulting Markov chain is always the same. It is also important to stress that this process of generating Markov chain stops after a finite number of steps because a transition occurs only once.

Now, in order to understand the whole process, from the specification to the generation of the Markov Chain through an example, consider a Standby Redundancy System that consists of a two machines (Machine 1 and Machine 2) where each machine is a standby for the other (VISWANADHAM; NARAHARI, 1988). Machine 1 has a higher priority to process a job when both the machines are idle and in working conditions. Machine 2 takes up the job whenever Machine 1 fails and simultaneously, Machine 1 is repaired. Yet Machine 2 may also break down while it is processing. In this case the job will be transferred to Machine 1 if it is in working condition. This model is shown in Figure 1 using Statecharts.

Figure 1 shows two parallel components Machine 1 and Machine 2. Both these components have three sub-states denoting Waiting (W1 and W2), Processing (P1 and P2) and Failure (B1 and B2). The initial state for Machine 1 is P1 whereas the initial default state for Machine 2 is W2. The events $\gamma_1$ and $\gamma_2$ indicate end of processing respectively for Machine 1 and Machine 2. A failure while processing is depicted using the events $\beta_1$ and $\beta_2$ whereas end of repair is indicated through the events $\mu_1$ and $\mu_2$. The figure 1 shows that Machine 1 has the priority to process a product and the Machine 2 takes over only when Machine 1 breaks and this fact is depicted in the figure making use of the true conditioned event (immediate event) $tr[in(B1)]$.

The reaction process starts by obtaining the initial configuration of the represented system, i.e. the initial states of all the parallel components. In the example shown in Figure 1 the initial states of the two components Machine 1 and Machine 2 are fetched. Therefore the initial configuration
yields \{P1,W2\}. The next step is to check whether there are any internal events that have to be stimulated immediately or not. In the example shown in Figure 1, no immediate events exist for the initial configuration. Now, a list of all the stochastic events (with a transition rate) is obtained, thus resulting in \( \beta_1 \) and \( \gamma_1 \). By stimulating each of these events from the list, a new configuration is obtained. For example, if \( \gamma_1 \) is taken to be stimulated, the generated configuration is \{W1, W2\}. However, as there is an internal event tr[not in(P2)] that is reacted immediately, this configuration automatically changes to \{P1, W2\}. Each new obtained configuration is kept in a list. Moreover, another data structure consisting of the source configuration, destination configuration, and the stochastic event that caused the change is also stored. This process goes on until a configuration is reached and its enabled events have already been stimulated. The state-transition diagram, i.e. the graph of configurations resulting from the reaction is shown in Figure 2.

This state-transition diagram has a one-to-one correspondence with a transition matrix where rows and columns correspond to the configurations and the elements of the matrix correspond to the transition rates. If the events in the transitions between states follow an exponential distribution, this graph is considered as a Markov Chain. The corresponding transition matrix of the Markov Chain of Figure 2 with the input rates is shown in Table 1. The elements of the matrix correspond to transition rates associated to the events \( \gamma_1(5.0) \), \( \gamma_2(5.0) \), \( \mu_1(3.0) \), \( \mu_2(3.0) \), \( \beta_1(0.1) \) and \( \beta_2(0.2) \). These rates have been arbitrarily provided in order to illustrate the input to be fed to the numerical methods for determining the steady-state probabilities.

![Figure 1: Standby Redundancy System](image)

<table>
<thead>
<tr>
<th></th>
<th>P1, W2</th>
<th>B1, P2</th>
<th>W1, P2</th>
<th>B1, B2</th>
<th>P1, B2</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1, W2</td>
<td>5.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>B1, P2</td>
<td>0.0</td>
<td>5.0</td>
<td>3.0</td>
<td>0.2</td>
<td>0.0</td>
</tr>
<tr>
<td>W1, P2</td>
<td>5.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.2</td>
</tr>
<tr>
<td>B1, B2</td>
<td>0.0</td>
<td>3.0</td>
<td>0.0</td>
<td>0.0</td>
<td>3.0</td>
</tr>
<tr>
<td>P1, B2</td>
<td>3.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.1</td>
<td>5.0</td>
</tr>
</tbody>
</table>

This transition matrix, which is a corresponding data structure to a Markov Chain, is the input
to numerical methods from which steady-state probabilities for each configuration can be obtained. Once these probabilities are obtained, they can be considered as performance measures due to their correspondence to the percentage of time that the model, during its dynamics, occupied a given configuration. Performance measures, based on the arbitrarily assigned transition rates (Table 1) associated to the events for the model of Figure 1, are shown in Table 2.

Table 2: Performance measures for the model of Figure 1

<table>
<thead>
<tr>
<th>Configurations</th>
<th>Performance measures</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1, W2</td>
<td>0.948475</td>
</tr>
<tr>
<td>B1, P2</td>
<td>0.030631</td>
</tr>
<tr>
<td>W1, P2</td>
<td>0.0176717</td>
</tr>
<tr>
<td>B1, B2</td>
<td>0.00105715</td>
</tr>
<tr>
<td>P1, B2</td>
<td>0.00216554</td>
</tr>
</tbody>
</table>

The model depicted in Figure 2 generated 5 configurations and 12 transition arcs. If another component, for instance, a Repairer (responsible for repairing the machines in case of failure) is added to the model, the number of generated nodes would be 6 with 14 transition arcs.

In a nutshell the process of generating steady-state probabilities of a given reactive system based on Statecharts specification can be characterized in the following steps:

1. Specify the model in Statecharts
2. Obtain the Markov Chain
3. Markov Chain has a one-to-one correspondence with a matrix structure. Lines and columns of this matrix are configurations (basic states of each orthogonal component). The elements of the matrix are transition rates.

4. Invoke the Markov Chain solution method by providing the transition matrix (obtained in 2) as input. Output of this process are the steady-state probabilities for each basic state of each component specified in (1).

This has been implemented within a tool called PerformCharts. Also included in the tool are probabilistic transitions as well as embedding memory into Markov models not considered within the scope of this paper.

The application of the specification technique Statecharts to performance models resulted in the development of a software package known as PerformCharts. In order to use PerformCharts, the specification of the model must be coded in the main module in C++ programming language followed by invoking the necessary methods to convert the representation into a Markov chain and then to calculate the performance measures. One natural alternative is a graphical interface to deal with PerformCharts, a language-based interface was used. XML (eXtensible Markup Language) has become very popular and extremely useful in dealing with interoperable formats (http://www.w3.org/XML/Activity). Therefore, a language based on this technology has been designed with the objective of its use in specifying and dealing with performance models. Features within XML have been transported to propose a markup language to deal with performance evaluation of reactive systems based on Statecharts specification. Therefore the markup language PerformCharts Markup Language PcML (AMARAL et al., 2004) has been proposed and developed. Its tags, attributes and other features represent the elements used in Statecharts for specifying reactive systems as well as their use in performance evaluation. Basically, following XML principles, tags consisting of the elements required to specify a complex system in Statecharts have been created for PcML. But the problem of handling the tags in order to generate the performance measures still remains. The solution to deal with this issue has been to interpret the language and generate the main module in C++ so that when compiled, linked with other classes from PerformCharts and executed would yield the performance measures. In other words, coding the specification and invoking necessary methods are thus avoided by letting the interpreter takes care of this. The interpreter has been written in two languages: perl and java.

4 Case Study

4.1 Manufacturing Cell With Robots

Imagine now a more complex system with a significant number of components to be represented such as the one shown in Figure 3 that depicts a Manufacturing cell with Robots already specified in Petri Nets by (VISWANADHAM; NARAHARI, 1988).

The cell with two identical machines $M_1$ and $M_2$, two material handling robots $R_1$ and $R_2$, three conveyors $C_1, C_2, C_3$, load and unload areas, is considered. Parts to be processed arrive at the left end of $C_1$ and reach the right end of $C_1$ through conveyor movement. Robot $R_1$ is responsible for moving them from $C_1$ to $C_2$. Parts are removed from $C_2$ by robot $R_2$ and are placed on $M_1$ or $M_2$ whichever is available. Processed parts are picked up by robot $R_2$ to place them on $C_3$ from which they are moved to the unloading area by robot $R_1$. Some assumptions considered for this example are:

1. Unprocessed parts are always available at the right end of $C_1$;
Figure 3: Manufacturing Cell with Two Robots

2. Each conveyor handles only one part at a time;
3. Machines, robots, and conveyors do not fail;
4. \( R_1 \) has priority to remove a part from \( C_3 \) to the unloading area over moving it from \( C_1 \) to \( C_2 \);
5. Robot transfer times and the machine processing times are exponentially distributed.

In Figure 3, machines, robots, and conveyors are modeled as parallel components. The substates and events in the Robot 1 component show that it moves parts from Conveyor 1 to Conveyor 2 besides moving them from Conveyor 3 to the Load Transfer Area. Conditions used in the event on the transition label from WR1 to LoadC2 guarantee a priority in moving a part from Conveyor 3 to the Load Transfer Area. Components of Machine 1 and Machine 2 consist of three sub-states ("Waiting", "Processing" and "Waiting to be Unloaded"). The component Robot 2 with its substates and events is responsible for moving parts from Conveyor 2 to either Machine 1 or Machine 2 besides unloading a part from either Machine 1 or Machine 2 to Conveyor 3.

However, the model represented in Statecharts is slightly different from the original Petri net representation described in (VISWANADHAM; NARAHARI, 1988). The difference is that when both the machines are idle, the robot has no way to decide whether to use Machine 1 or Machine 2. In Petri nets a probability of 0.5 was assigned and here in Statecharts a priority to Machine 1 was provided.

This example, when generating the state-transition diagram, yielded 44 tangible states and 80 transitions (VIJAYKUMAR et al., 2002). Input parameters for this model are shown in Table IV while the performance measurements are given in Table V. A state-transition diagram with this number of states and arcs depicting the behavior of a model is not easy to be constructed. Even though one manages to construct such a diagram, the behavior of the model cannot be easily understood.
Table 3: Input Parameters for the model of Figure 3

<table>
<thead>
<tr>
<th>Transition rates</th>
<th>Mean part transfer time from C1 to C2</th>
<th>40 (EndLoadC2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean part transfer time from C2 to M1</td>
<td>20 (EndLoadM1)</td>
</tr>
<tr>
<td></td>
<td>Mean part transfer time from C2 to M2</td>
<td>20 (EndLoadM2)</td>
</tr>
<tr>
<td></td>
<td>Mean processing time on M1</td>
<td>1 (EndPM1)</td>
</tr>
<tr>
<td></td>
<td>Mean processing time on M2</td>
<td>1 (EndPM2)</td>
</tr>
<tr>
<td></td>
<td>Mean part transfer time from M1 to C3</td>
<td>20 (EndLoadC3)</td>
</tr>
<tr>
<td></td>
<td>Mean part transfer time from M2 to C3</td>
<td>20 (EndTransfer)</td>
</tr>
</tbody>
</table>

Table 4: Performance measurements for the model of Figure 3

<table>
<thead>
<tr>
<th>Performance measurements</th>
<th>Utilization of M1</th>
<th>0.9002</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Utilization of M2</td>
<td>0.8948</td>
</tr>
<tr>
<td></td>
<td>Utilization of R1</td>
<td>0.0887</td>
</tr>
<tr>
<td></td>
<td>Utilization of R2</td>
<td>0.1814</td>
</tr>
</tbody>
</table>

looking at it. This is the reason that a use of sophisticated high-level specification methods such as Petri nets or Statecharts is very much justified.

Example covered in Figure 3 originally had some interesting aspects to be considered. In the original specification (VISWANADHAM; NARAHARI, 1988), some conflicts could have occurred. For example, when a part was ready to be processed either by Machine 1 or Machine 2 and both were idle, the robot could not decide to which machine the part was to be assigned. The original representation in Petri nets solved this situation by introducing probabilities in the transitions. In the case specified by Statecharts, priorities were assigned as can be seen in Figure 3.

5 Final Remarks

Specification of complex systems is not an easy task and it is not the intention of this paper to claim that Statecharts are the best option for all the possible situations that can appear in modern complex systems. However, the potential features provided in Statecharts open a wide range of intricacies to be specified in the model. Examples shown here and others tested seem to fall into the category where Statecharts are a better option when compared to other specification techniques. One more interesting factor is that Statecharts are a natural extension of state-transition diagrams that are very closely associated to Markov Chains, which are very much used in analytical solutions to determine performance measures.

Reaction of events occur in parallel. However the algorithm puts the enabled events in a sequence. Usually the order of the sequence is the same used in the specification. It is important to stress that the order of the events does not interfere in the generation of the Markov chain which is unique.

Interface has been significantly improved by providing XML-based language to specify a reactive system in Statecharts. Scripts written in perl and Java deal with the language to generate the main program in C++ code. Graphical interface is the ideal one and it is under development. Web-based PerformCharts is also under development. This very same tool has been adapted to deal with test sequence generation. This has been decided as several test sequence generation techniques are available in the literature that are used when a software specification is given in a finite state machine whose graphical correspondence is a state-transition diagram. Therefore, PerformCharts
generate a corresponding state-transition diagram of a Statecharts representation and this has been extensively used on in-house software for space applications.

Acknowledgements
The authors wish to acknowledge CAPES that has provided financial support within PROCAD program under the project 0226/05-0.

References


