On proposing a Markup Language for Statecharts to be used in Performance Evaluation

ANA SILVIA MARTINS SERRA DO AMARAL
Laboratory of Computing and Applied Mathematics (LAC)
National Institute for Space research (INPE)
BRAZIL

RENÊ RODRIGUES VELOSO
IBM Brazil/Acesita
BRAZIL

N. L. VIJAYKUMAR
Laboratory of Computing and Applied Mathematics (LAC)
National Institute for Space research (INPE)
BRAZIL
Coastal and Marine Resources Centre (CMRC)
University College Cork(UCC)
Naval Base, Haulbowline, Cobh, Co. Cork
IRELAND

Abstract: - Statecharts have been originally created to specify complex reactive systems for use in simulating real-time applications. They are an extension of state transition diagrams with notions of hierarchy, concurrency and synchronization. Recently they have been used for specifying and dealing analytically with performance models in which a Statecharts representation is converted into a Markov chain from which steady-state probabilities are obtained. A software system, PerformCharts, has been developed for this purpose. At the moment, the specification of a reactive system and its conversion into a Markov Chain, within PerformCharts, have to be written in C++ language within the main program. The paper, therefore, proposes a markup language based on XML to serve as an interface to specify, in Statecharts, a complex reactive system with the objective of evaluating its performance. Procedures of how to deal with such specification in order to generate the necessary performance evaluation information are provided.

Key-Words: - Statecharts, PcML, Performance, Markup, XML

1 Introduction
Behavior of a system can be studied by means of combining modeling with performance evaluation process. Modeling now has become an important area of research as the present systems to be evaluated deal with complex logic that have to be represented. The scope of this research concentrates on systems that are known to be complex as they are reactive. Reactive systems are those that their behavior changes at a given instant according to a perturbation. The scope of this paper is concentrated on Markov chain-based analytical approach. Markov chains can usually be expressed in state-transition diagrams that consist of states and transition arcs between states and, the events among the states must follow an exponential distribution. However, the use of state-transition diagrams may be cumbersome especially when dealing with parallel components as they may lead to exponential blow-up of the model [1]. The work described here, PerformCharts, is based on Statecharts representation [2-6] of a complex reactive system and this representation is converted into a Markov chain [7,8] in order to obtain performance measures. The procedure of exploring Statecharts in performance evaluation begins by specifying a
reactive system. However, so far no user-friendly interface (either graphical or textual) has been developed for PerformCharts. Specification of a given reactive system has to be embedded in the main program where the data structures are constructed to generate a Markov chain. Therefore this paper proposes a markup language, PcML (PerformCharts Markup Language), based on XML (eXtensible Markup Language) for specifying a complex reactive system. The problem still continues on how to deal with this markup language so that PerformCharts can be run to generate performance measures. Two approaches (one based on Java and the other based on perl) are provided where the specification written in PcML is converted into the main program from which the developed software can be linked and run thus generating the performance measures. The paper is organized as follows: Section 2 provides a brief description of how a Markov chain is generated from a Statecharts representation. Section 3 addresses basic concepts of XML and introduces the proposed markup language for PerformCharts. Section 4 presents the ideas on how to deal with PcML and some of the operators constructed in order to obtain the performance values for the specified reactive system. Finally Section 5 ends the paper with some conclusions.

2 Statecharts and their use in Performance Models

Statecharts extend state-transition diagrams with notions of hierarchy (depth), orthogonality (parallel activities) and interdependence/synchronization (broadcast communication).

Definitions of elements that form the basis of Statecharts technique as well as the main features are described in [2], [3], [5] and [6].

One of the main elements of Statecharts (besides states, conditions, actions, transitions), events, play a major role as they are responsible for changing the behavior of a system. As they are crucial in the scope of performance evaluation, their brief discussion is in order [7] and [8]. Events have been classified into two categories: internal and external. Internal events are those that take zero time when they are enabled.

Also known as immediate events they are triggered automatically by the internal logic of Statecharts, i.e., they are not explicitly stimulated. Statecharts have some built-in events such as true(condition), false(condition), entered(State), exit(State). The basic element action when used as an event that can influence some other orthogonal component is also considered as an immediate event within the scope of this paper. This means that whenever an event is associated as action, a reaction to this event is immediate. External events are stochastic events (where time between their activation and their occurrences follow a stochastic distribution) that have to be externally stimulated to yield new configurations. In order to make the association of a Statecharts model with a Markov chain, the only type of events considered are stochastic events. In particular, for Continuous-Time Markov Chains, this distribution has to be exponential. Based on an initial configuration (basic states of each orthogonal component in the initial instant), enabled events are listed. By considering each event from the list, a reaction is performed, yielding a new configuration for each event of the list. Once a configuration is obtained, internal events, if enabled, are triggered, firing transitions to yield new configurations. This process continues until all the configurations have been expanded. The result of all this process is a list of a structure that contains a source configuration, stimulated stochastic event (along with its transition rate), and the target configuration. This information is a Markov chain with which steady-state probabilities can be determined [9] and [10]. In order to understand the specification of a reactive system using Statecharts, consider an example of a System with two machines and a repairer shown in Fig.1 and its resulting Markov Chain in Fig.2.

The initial configuration is obtained by taking the default basic states of each parallel component E1, E2 and Supervisor, i.e., [W1,W2,WS]. First of all, the algorithm to generate the Markov chain looks for any enabled immediate events. There are no such events for the initial configuration.
The only enabled events are the stochastic events $a_1$ and $a_2$. Therefore, these events will be stimulated to yield the new configurations. By stimulating $a_1$, the configuration is $[P_1,W_2,WS]$ and with $a_2$ $[W_1,P_2,WS]$. This process goes on and for each new configuration immediate events are checked if they are enabled. Just as an example, at some stage if the configuration is $[B_1,W_2,WS]$, the enabled immediate event is $tr_{in}(B_1)$ and a transition is fired by immediately moving $WS$ to $C_1$. This reaction process terminates when a configuration is reached such that its list of enabled stochastic events has already been triggered. The resulting Markov chain is shown in Figure 2. A numerical method [9] and [10] is used to solve the Markov chain and steady-state probabilities are resulted and they are the basis for the performance measures.

3 PcML - PerformCharts Markup Language

XML is a language consisting of a set of rules that can be used for formatting texts in order to structure a given data. The main advantage of XML is that it is extensible, platform-independent, and it is supported by international efforts for standardization. XML is similar to HTML by making use of tags and attributes. In HTML, tags and associated attributes are used to present the text in a certain format by a browser. Whereas in the case of XML, tags are used to organize a set of data and the interpretation of this data is entirely left to the application that eventually reads it. It is just more than a markup language. It can be used to model the contents of a document, i.e., a standard for the contents. By using regular text format, XML makes it possible for any reader to look at the text without having to use any program for that purpose, i.e., one can use any text editor to read XML file. However, XML is strict in the sense that a forgotten tag or an attribute value without quotes generates an error [11]. These features within XML have been transported to propose a markup language to deal with performance evaluation of reactive systems based on Statecharts specification. PcML [12] is a markup language, based on XML, whose tags, attributes and other features represent the elements used in Statecharts for specifying reactive systems as well as their use in performance evaluation [13] and [14]. A diagram of the PcML is shown in Fig.3.

![Fig.3: PcML Diagram](image)

Fig.3 shows boxes consisting of elements using dotted and solid lines. Elements with dotted lines are not required in the specification of some systems while elements with solid lines are mandatory. This is because some of the elements, such as conditions, actions and probabilities are optional whereas states, events and transitions are mandatory for specifying a complex reactive system. The optional elements provide more power in representing complex systems. The boxes connected to other boxes, with
the symbol -> are the elements allowed between the matching start and end tags. Consider the element Conditions; the tags allowed between it and /Conditions are: InState, NotCondition and ComposedCondition. In the same way, the elements ANDCond and ORCond can be specified within ComposedCondition.

The $1,\infty$ means that the number of occurrences of such elements must be in the range of one to infinite.

XML 1.0 supplies the Document Type Definition (DTD) mechanism for declaring constraints on the use of markup [15].

**4 On dealing with PcML for performance evaluation**

The specification part within PerformCharts’ main program is basically function calls for the creation of the states, conditions, events, actions, transitions and other elements of a reactive system.

Following are some details on how those elements are defined within PcML file.

1. **Root**
   A Root element must be specified for the model in Statecharts and to which performance measures are to be calculated. In the case of the example shown previously in Figure 1, the definition in PcML file is:
   
   ```xml
   <Root Name="System" Type="AND"/>
   ```

   The RootName corresponds to the name given to the model.

2. **States**
   In this step, states (root as well as its offspring) are created defining the whole hierarchy of the specified model. As mentioned elsewhere, an initial default state must be provided. In case, entry by History is considered, this information must also be given.

   While creating a root state, a string containing the name assigned to this root state and its type have to be provided as parameters. For the rest, a name, its type, its parent have to be provided. Definition of default states needs the state that should be the entry point as well the component in which this default is applied must be informed. Taking the example of Figure 1, PcML defines states as shown below:
   
   ```xml
   <State Name="E1" Type="XOR" Default="W1"/>
   ```

3. **Conditions**
   Conditions associated with events affect transitions. If they are not satisfied a transition is not fired even though the event associated to them is enabled. Conditions can also be combined.

   As an example, a frequently used condition - instate condition - in[X], where X is a State is defined in the following way:
   
   ```xml
   <InState Name="C1" State="X"/>
   ```

   The definition indicates that a instate condition named C1 has been created and the related state has the name X. The State must have already been defined before.

4. **Events**
   Recalling the category of events mentioned earlier, there are internal (immediate) events (built-in and actions) and external (stochastic) events. Events without stochastic information are considered as actions, i.e., when an event is triggered, a transition is fired and if there is an action associated to the event, reaction continues by changing the system configuration.

   When defining a stochastic event, a name (f1) for the event along with its transition rate (0.1) must be specified. In case an immediate (built-in) event is needed, for example a true condition event, the tag TrueCondition is used by specifying a name (TC1) for the event along with the condition (C1). The condition must have been defined earlier. Examples of such events are listed:
   
   ```xml
   <Events>
   <Stochastic Name="f1" Value="0.1"/>
   <TrueCondition Name="TC1" Condition="C1"/>
   </Events>
   ```

   In order to define Action, EventTriggerAction must be used with the action name (Eta1M1) along with the associated event (c1). Example is listed below:
   
   ```xml
   <Actions>
   <EventTriggerAction Name="Eta1M1" Event="c1"/>
   </Actions>
   ```

5. **Transitions**
   Transitions are arcs joining source states to target states. Labels on these arcs are events or actions for moving the present state to another. In order to define a transition within PerformCharts, the following are required: a name for a transition; means by which the transition occurs (event, conditioned event, action, etc.); source state; and a target state. From the example in Figure 1, a transition in PcML is defined as:
   
   ```xml
   <Transition Source="W1" Event="a1" Destination="P1"/>
   ```

   Once the PcML file is ready, somehow it has to be manipulated in order to generate a Markov chain from the Statecharts model and also to determine the performance measures. Therefore, the solution consists in parsing PcML file and the information contained within is converted into a main program in C++. This generated main program duly linked to other classes and run will generate the required output. The PcML file is interpreted by using two approaches: one in perl and the other in java. Both the implementations access and manipulate the PcML
document by means of DOM (Document Object Model)-based parsing. This parser reads an XML document and creates objects to represent the different parts of that document. These objects are associated with specific methods and properties, and are used to manipulate and access information about it. Thus, the entire XML document is represented as a hierarchy "tree" of these objects, with the DOM parser providing a simple API to move between different branches. Once a particular node has been reached, built-in methods can be used to obtain value of the node, and use it within the script [16]. The DOM specification treats every part of the document as a node consisting of a type and a value. It supports all the different structures typically found in an XML document: elements, attributes, namespaces, entities, notations and others. The DOM specification is designed to be usable with any programming language. Therefore, it attempts to use a common core set of features which are available in all languages: DOM defines a standard set of interfaces for representing documents, a standard model of how these objects can be combined, and a standard set of methods for accessing and manipulating them. The DOM specification also attempts to remain neutral in its interface definitions. DOM is a W3C Recommendation and it is recognized as a Web standard. Both programs read the PcML file and, according to the rules established in a DTD/Schema, check if it is well formed.

Once the PcML is converted into a main program, it is linked to the appropriate library to generate PerformCharts executable file to be run and to generate performance measures. Perl code consists in traversing the PcML document searching for given tags with their values and attributes. Once retrieving these tags along with their corresponding values and attributes, it writes text lines in the main program file consisting of functions calls of PerformCharts.

Detailed information on how to use XML in Perl, can be found in [13] and [17].

In Java, after the hierarchy tree of nodes/objects is created, the nodes will be searched by internal methods which use Xpath expressions. Detailed information on how to use XML in Java and XPath, can be found in [14] and [18].

Now, some examples of C++ commands generated by those tools are in order.

**Statechart System:**
- System.createRoot("System",AND);
- System.createSonState ("E1",OR,"System");
- System.setDefaultEntry ("E1","P1");

One can observe that the name assigned to the root state is also assigned to the system being modeled.

The example shown in Figure 1 does not use any entry by History feature. In the case it were used, a command line similar to:
- System.createHistoryEntry("State") ;

were generated.

Next, an example of the creation of a condition:
- InStateCondition *inC1 = System.createInStateCondition("W1");

In the case of events, definition of stochastic events is based on the method createPrimEvent and its parameters are a string with the event’s name and a transition rate. The same method is used without the transition rate for defining an event that will be used as an action.

Just recalling the specification of an action for clarification purposes, in C++, action is first defined as a primitive but without any value for the transition rate. Only then, this primitive event is defined as an action through the class EventTriggerAction. It has the same semantic meaning as pre-defined internal events (true, false, entered, exited) when considering the dynamics of Statecharts. In the case of pre-defined internal events they are also identified by names. However, some restrictions are in order: in case of true and false, if they are related to conditions, these conditions must have already been defined; when considering entered and exited, they use a state as their parameter, and the state must have already been defined. An example for True Condition Event is shown. Remember that the Condition C1 must have been defined earlier:
- System.createTrueCondEvent (inC1);

In Statecharts any event can be combined with a condition in order to guard the event meaning that even the event is enabled, it cannot be triggered if the guarding condition is not satisfied. Therefore, a class ConditionedEvent has been designed to deal with such situations and this class takes two parameters, one for the event and the second is the condition. The example shown in the Figure 1 has no such events.

**ConditionedEvent *cevR1= example.createConditionedEvent (*ncR1, "Lambda r");

Example of the creation of a transition are followed:
- Transition *transW1_P1 = System.createTransition (*teva1);
  - System.addSourceNode (*transW1_P1,"W1");
  - System.addDestinationNode (*transW1_P1,"P1");

Finally, the following C++ function calls are generated in order to yield the performance evaluation parameters:
- GraphBase gb;
- GraphGenerator gen;
  - //Perform reaction - generate all possible Configurations
5 Conclusion
Complex reactive systems are one of a kind where many intricacies have to be represented. Statecharts’ powerful features make it easier to represent such systems in most of the cases. For performance models, Statecharts representation was associated to a mathematical solution, in particular to Markov chains from which performance measures are obtained. The question remained was the interface. A graphical interface is being developed. However, due to the growing use of interoperability technologies, especially XML, it has been decided to adopt it in the context of performance evaluation. Thus, by adapting XML to PerformCharts, PcML was created. The development of such language along with the solution approaches to deal with it were quite fast and started as a course project. This interface gave a boost to the use of PerformCharts due to its easiness in specifying a performance model. Further work related to this project is to develop a web-based PerformCharts where a reactive system can be delivered in graphical form or in PcML in order to calculate the performance measures of the given system. The main idea here would be to convert the graphical interface to PcML. However, other approaches of generating the performance measures without having to convert into a main program are under consideration.

References: