



MINISTÉRIO DA CIÊNCIA E TECNOLOGIA
INSTITUTO NACIONAL DE PESQUISAS ESPACIAIS

INPE-13260-RPQ/799

**ON PROPOSING A MARKUP LANGUAGE FOR STATECHARTS
TO BE USED IN PERFORMANCE EVALUATION**

Ana Silvia Martins Serra do Amaral
René Rodrigues Veloso
Nandamudi Lakalapalli Vijaykumar

Relatório Técnico.

ABSTRACT

Statecharts have been originally created to specify complex reactive systems for use in simulating real-time applications. They are a graceful 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. The software system that specifies a reactive system in Statecharts and generates the performance measures is known as PerformCharts. At the moment, specification of a reactive system, within PerformCharts, is done in the main program written in C++ programming language. The objective of this paper is to propose a markup language based on XML to specify a complex reactive system as well as to comment procedures used to deal with this language in order to obtain performance information.

PROPOSTA DE UMA LINGUAGEM DE MARCAÇÃO PARA STATECHARTS PARA SER USADA EM AVALIAÇÃO DE DESEMPENHO

RESUMO

Statecharts foram, originalmente, criadas para especificação de sistemas reativos complexos para uso em simulação de aplicações de tempo real. São uma elegante extensão de diagramas de transição de estados com noções de hierarquia, concorrência e sincronismo. Recentemente, têm sido utilizados para especificação e tratamento analítico de modelos de desempenho onde a representação Statecharts é convertida numa cadeia de Markov de onde as probabilidades limite são obtidas. O software que especifica um sistema reativo em Statecharts e gera suas medidas de desempenho é chamado de PerformCharts. Atualmente, a especificação de um sistema reativo, dentro do PerformCharts, é feita dentro de um programa principal na linguagem C++. O objetivo deste trabalho é propor uma linguagem de marcação baseada em XML para a especificação de um sistema complexo reativo como também comentar os procedimentos para o tratamento dessa linguagem a fim de obter informação de desempenho.

CHAPTER 1

INTRODUCTION

In general, behavior of a system can be studied by modeling and evaluating its performance. It is possible to detect some bottle-necks during the evaluation process. Modeling now has become an important area of research as the representation of a given system allows an insight of system behavior before it is constructed. The scope of this research work 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. In order to evaluate the performance of a given system, two categories of solutions are used - simulation and analytical approach. The scope of this paper is concentrated on solutions via analytical methods based on Markov theory. The first natural solution of representing a reactive system is state-transition diagrams. This is due to the fact that Markov chains are usually expressed in these diagrams that consist of states and transition arcs between states. If the events among the states follow an exponential distribution, then this structure may be considered as a Markov chain. Once a system is represented as a Markov chain, by applying appropriate numerical methods (PHILIPPE et al., 1992) (SILVA, 1992), one can determine the steady-state probabilities with which performance measures can be obtained. However, the use of state-transition diagrams may be cumbersome especially when dealing with parallel components as they may lead to exponential blow-up of the model (DRUSINSKY e HAREL, 1989). The work described here, PerformCharts, is based on Statecharts representation (HAREL, 1987; HAREL et al., 1987, 1990; HAREL e NAAMAD, 1996; HAREL e POLITI, 1998) of a complex reactive system and this representation is converted into a Markov chain (VIJAYKUMAR, 1999; VIJAYKUMAR et al., 2002). The procedure exploring Statecharts in performance evaluation begins with the specification of a reactive system. The main problem is that so far no user-friendly interface (either graphical or textual) has been developed for PerformCharts. This means that the specification of a given reactive system has to be embedded in the main program where the data structures are constructed in order to deal with the generation of 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 the software is run in order to generate performance measures. Two approaches (one based on Java and the other based on perl) will be commented where the specification written in PcML is converted into the main program from which the developed software can be run and consequently generating the performance measures.

1.1 OUTLINE

The paper is organized as follows:

- SECTION 2 - Statecharts and their use in Performance Models: provides a brief description of Statecharts as well as how a Markov chain is generated ;
- SECTION 3 - XML - eXtensible Markup Language: addresses basic concepts of XML.
- SECTION 4 - PcML - PerformCharts Markup Language: introduces the proposed markup language for PerformCharts .
- SECTION 5 - On dealing with PcML for Performance Evaluation: presents the Java based as well as perl based approaches in order to generate the main program.
- SECTION 6 - CONCLUSIONS: comments and future plans are discussed.

CHAPTER 2

Statecharts and their use in Performance Models

Statecharts are graphical-oriented and are capable of specifying reactive systems. They have been originally developed to represent and simulate real time systems (HAREL, 1987). Moreover Statecharts come with a strong formalism (HAREL et al., 1987) (HAREL e 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 when there are no further refinements from this type of state. In Statecharts the global state of a given model is referred to as a configuration that is the active basic states of each orthogonal component. Details of definition of each element as well as the main features are described in (HAREL, 1987), (HAREL et al., 1987), (HAREL e NAAMAD, 1996) and (HAREL e POLITI, 1998). By definition, when modeling a given system, there must always be an initial state also known as default state in Statecharts. When dealing with orthogonal components, it is more natural to use initial or default configuration indicating the initial or default states within each orthogonal component. 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. A brief discussion of events considered for performance evaluation (VIJAYKUMAR, 1999) and (VIJAYKUMAR et al., 2002) is in order. Events have been classified into two categories: internal events and external events. Internal events are those that take zero time when they are enabled. They are also known as immediate events as the reaction to these take place immediately. They are triggered automatically by the in-

ternal logic of Statecharts, i.e, they do not have to be explicitly stimulated. Statecharts have such built-in events: 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 for the purposes of applications within performance evaluation. 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 (which consists of converting the Statecharts specification into 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. Once the model is specified in a Statecharts representation, 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 resulting configurations, reactions continuously take place by changing one configuration to another until a configuration is reached from which no more internal events are active to be triggered. The next step is to deal with the stochastic events that have to be explicitly stimulated. Therefore, the algorithm, based on the resulting configuration from internal events, lists which stochastic events are to be triggered so that transitions are fired to generate new configurations. By considering each event from the list, a reaction is performed, yielding a new configuration for each event of this list. 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. The result of all this process is a list of a structure that contains a source configuration, stimulated stochastic event (along with its rate), and the target configuration. This information is a Markov chain with which steady-state probabilities can be determined. The whole process through an example is described in (VIJAYKUMAR et al., 2002).

CHAPTER 3

XML - eXtensible Markup Language

XML is a kind of meta language, that is, it is not a programming language and can be considered as 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. Just as an example, while `<p>` in HTML indicates a paragraph, in XML it might inform price, parameter, person, etc. depending on the context. 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. XML has a set of useful modules to accomplish some tasks such as: Xlink allows to add hyperlinks to an XML file; Xpointer and Xfragments are useful in pointing to parts of an XML document; CSS is a style sheet language, just like for HTML, to apply to XML files. XSL is a language that can describe style sheets to enable to rearrange, add and delete tags and attributes; DOM allows handling XML files from a programming language; Schemas assist users to define structures of XML-based formats. The best way to appreciate and get a feeling of what XML documents look like is with a simple example - a company that sells products on-line. HTML formatting is used to describe the products; however, names and addresses of customers, and also prices and discounts are formatted with XML. A customer is described as:

```
<customer-details id="AcPharm39156">
<name>Acme Pharmaceuticals Co.</name>
<address country="US">
<street>7301 Smokey Boulevard</street>
<city>Smallville</city>
<state>Indiana</state>
<postal>94571</postal>
</address>
</customer-details>
```

All the tags must have a matching end within XML syntax. As can be seen from the example, start and end tags `<name>` and `</name>` are used to mark up informa-

tion. Information marked by the tags is called an element; elements may be further enriched by attaching name-value pairs (for example, country="US" in the example above) called attributes. Its syntax is quite simple syntax and can be easily processed by a machine. Moreover, it is understandable to humans. XML is based on SGML, and has a lot of similarities when compared to HTML. Detailed information about XML can be found in (Consortium, 2002c; BATES, 2003; BONNEAU et al., 2003; GRAHAM, 1999; HOLZNER, 1998; MADEN, 2000; MADEN e RAY, 2001; PINNOCK et al., 2001; ST. LAURENT, 1999; YOUNG, 2001; Iso, 2002).

CHAPTER 4

PcML - PerformCharts Markup Language

PcML 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. A diagram of the PcML is shown in Figure 4.1.

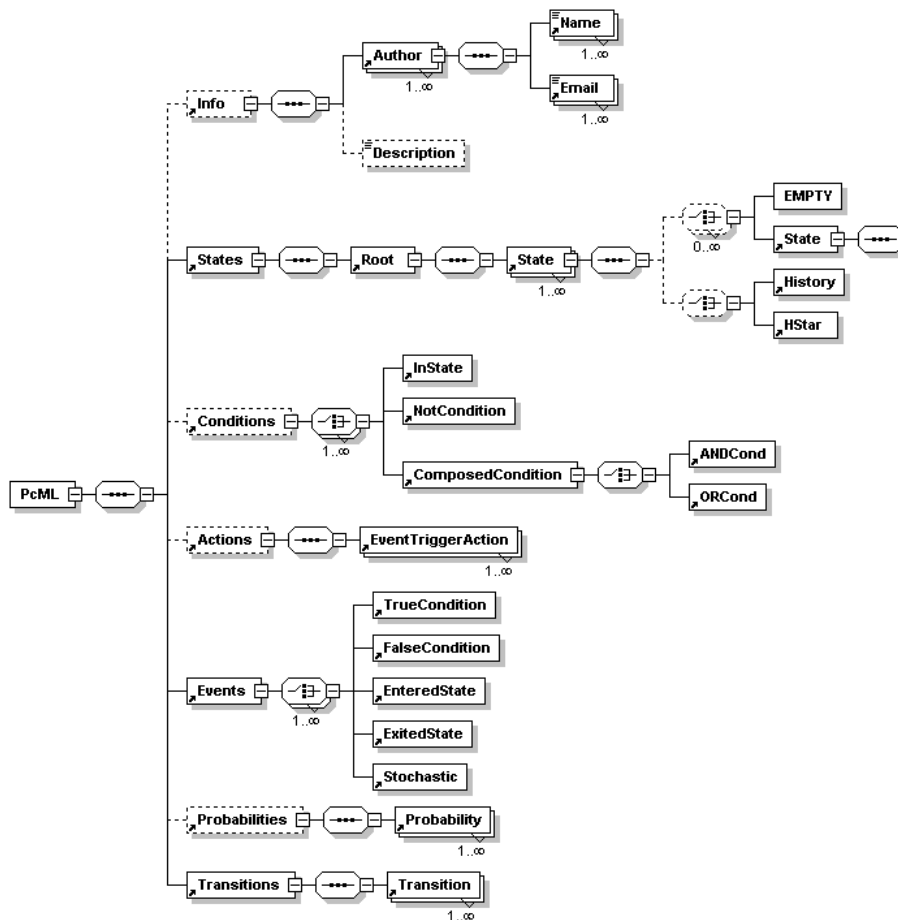


FIGURE 4.1 – PcML Diagram.

In the figure 4.1, one can notice boxes using dotted and solid lines. The former ones are not required in the specification of some systems while the latter ones are mandatory. This is because some of the elements from Statecharts are optional, such as: conditions, actions and probabilities, in contrast to the elements such as: states, events and transitions, that

always must be used for specifying a system in Statecharts. For instance, the element description of info serves merely for documentation purposes while the element States will be used in the calls of functions in PerformCharts. 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 are allowed to be specified within ComposedCondition, and so on. The 1.. ∞ means that the number of occurrences of such elements must be in the range of one to infinite. The general structure of a PcML file is shown in Figure 4.2.

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<PcML xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="PcMLschema.xsd" Title="" Date="">
  <Info>
    <Author>
      <Name/>
      <Email/>
    </Author>
    <Description/>
  </Info>
  <States>
    <Root Name="" Type="">
      <State Name="" Type="" Default="">
        <EMPTY/>
        <History/>
      </State>
    </Root>
  </States>
  <Conditions>
    <InState Name="" State=""/>
    <NotCondition Name="" Condition=""/>
    <ComposedCondition Name="">
      <ANDCond Cond1="" Cond2=""/>
    </ComposedCondition>
    <ComposedCondition Name="">
      <ORCond Cond1="" Cond2=""/>
    </ComposedCondition>
  </Conditions>
  <Actions>
    <EventTriggerAction Name="" Event=""/>
  </Actions>
  <Events>
    <TrueCondition Name="" Condition=""/>
    <FalseCondition Name="" Condition=""/>
    <ExitedState Name="" State=""/>
    <EnteredState Name="" State=""/>
    <Stochastic Name="" Value=""/>
  </Events>
  <Probabilities>
    <Probability Name="" Value=""/>
  </Probabilities>
  <Transitions>
    <Transition Source="" Event="" Condition="" Probability="" Action="" Destination=""/>
  </Transitions>
</PcML>

```

FIGURE 4.2 – General Structure of a PcML file.

XML 1.0 supplies the Document Type Definition (DTD) mechanism for declaring constraints on the use of markup, but automated processing of XML documents requires

more rigorous and comprehensive facilities in this area. Requirements are for constraints on how the component parts of an application fit together, the document structure, attributes, data-typing, and so on. The XML Schema Working Group is addressing means for defining the structure, content and semantics of XML documents (Consortium, 2002a). The purpose of a schema is to define and describe a class of XML documents by using these constructs to constrain and document the meaning, usage and relationships of their constituent parts: datatypes, elements and their content, attributes and their values, entities and their contents and notations. Schema constructs may also provide for the specification of implicit information such as default values. Schemas document their own meaning, usage, and function. Thus, the XML schema language can be used to define, describe and catalogue XML vocabularies for classes of XML documents (Consortium, 2002b). For more details about the Schema for PcML, please refer to appendix A. In order to understand the specification of a reactive system within PcML consider an example of a System with two machines and a repairer shown in Figure 4.3.

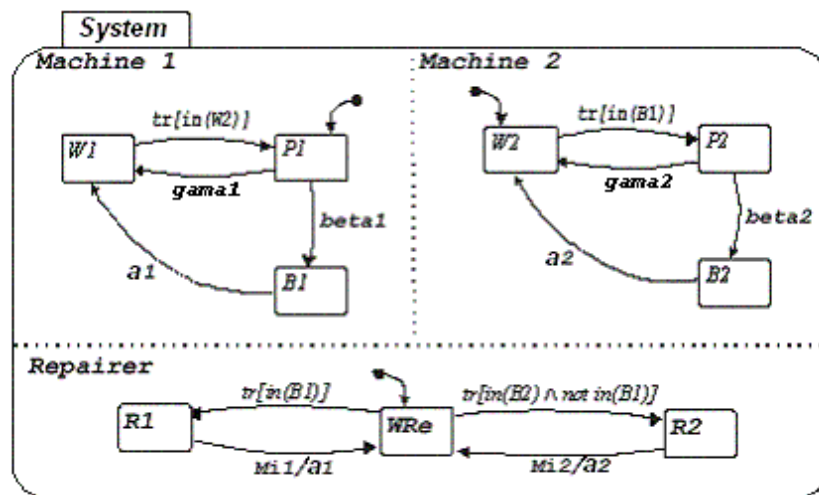


FIGURE 4.3 – Specification of a system with two machines and one repairer. This system is comprised of two Machines and one Repairer. The component repairer is responsible for repairing the machines when they fail and a priority is provided to machine 1 whenever both the machines are down. The behavior of the machines and the repairer is modeled by the state transition diagram, where the basic states in both the machines are: W (Waiting), P (Processing) and B (Broken). For the repairer, the basic states are: R1 (Repairing Machine 1), WRe (Waiting) and R2 (Repairing Machine 2).

<i>Transition</i>	<i>Source</i>	<i>Event</i>	<i>Condition</i>	<i>Action</i>	<i>Destination</i>
1	W1		in(W2)		P1
2	P1	gama1			W1
3	P1	beta1			B1
4	B1	a1			W1
5	W2		in(B1)		P2
6	P2	gama2			W2
7	P2	beta2			B2
8	B2	a2			W2
9	WRe		in(B1)		R1
10	R1	Mi1		a1	WRe
11	WRe		n(B2)^not in(B1)		R2
12	R2	Mi2		a2	WRe

FIGURE 4.4 – Transitions of the system represented in Figure 4.3.

The initial configuration of the system is obtained by taking the default basic states of each parallel component Machine 1, Machine 2 and Repairer, i.e., (P1,W2,WRe). The PcML specification of such reactive system is given in the Figure 4.

Once the specification is over, a first reaction is performed by first checking the internal events. In this example the internal events are: $\text{tr}[\text{in}(W2)]$, $\text{tr}[\text{in}(B1)]$, $\text{tr}[\text{in}(B2) \wedge \text{not in}(B1)]$. The initial configuration has an active state W2, so the internal event $\text{tr}[\text{in}(W2)]$ is enabled and the configuration becomes (P1,W2,WRe). Next, the enabled events are the stochastic ones. In the list, one can notice the stochastic events: gama1 and beta1. In the case of the event gama1 were enabled, the next configuration would be (W1,W2,WRe) or, if beta1 were enabled the next configuration would be (B1,W2,WRe). Suppose that during the process of stimulating the events, a configuration is (B1,B2,WRe). In this case, the active events are the immediate events provided by $\text{tr}[\text{in}(B1)]$ and $\text{tr}[\text{in}(B2) \wedge \text{not in}(B1)]$. These are the events that have to be checked and enabled before the stochastic events.

```

<?xml version="1.0"?>
<PerformCharts title="Standby Redundancy System" date="2002-12-01"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="PcMLschema.xsd">
  <Info>
    <Author>
      <Name>Ana Silvia</Name>
      <E-mail>anasil@lac.inpe.br</E-mail>
    </Author>
  </Info>
  <States>
    <Root Name="System" Type="AND">
      <State Name="Machine 1" Type="XOR" Default="P1">
        <State Name="P1" Type="BASIC"/>
        <State Name="W1" Type="BASIC"/>
        <State Name="B1" Type="BASIC"/>
      </State>
      <State Name="Machine 2" Type="XOR" Default="W2">
        <State Name="P2" Type="BASIC"/>
        <State Name="W2" Type="BASIC"/>
        <State Name="B2" Type="BASIC"/>
      </State>
      <State Name="Repairer" Type="XOR" Default="WRe">
        <State Name="WRe" Type="BASIC"/>
        <State Name="R1" Type="BASIC"/>
        <State Name="R2" Type="BASIC"/>
      </State>
    </Root>
  </States>
  <Conditions>
    <InState Name="C1" State="W2"/>
    <InState Name="C2" State="B1"/>
    <InState Name="C3" State="B2"/>
    <NotCondition Name="C4" Condition="C2"/>
    <ComposedCondition Name="C5">
      <ANDCOND Cond1="C3" Cond2="C4"/>
    </ComposedCondition>
  </Conditions>

```

FIGURE 4.5 – PcML Specification of the system represented in Figure 4.3.
(continue)

```

<Actions>
  <EventTriggerAction Name="Eta1M1" Event="a1"/>
  <EventTriggerAction Name="Eta1M2" Event="a2"/>
</Actions>
<Events>
  <TrueCondition Name="E1" Condition="C1"/>
  <TrueCondition Name="E2" Condition="C2"/>
  <TrueCondition Name="E3" Condition="C5"/>
  <Stochastic Name="Beta1" Value="0.1"/>
  <Stochastic Name="Gama1" Value="5.0"/>
  <Stochastic Name="Beta2" Value="0.2"/>
  <Stochastic Name="Gama2" Value="5.0"/>
  <Stochastic Name="Mi1" Value="3.0"/>
  <Stochastic Name="Mi2" Value="3.0"/>
</Events>
<Transitions>
  <Transition Source="W1" Event="E1" Destination="P1"/>
  <Transition Source="P1" Event="Beta1" Destination="B1"/>
  <Transition Source="P1" Event="Gama1" Destination="W1"/>
  <Transition Source="B1" Event="a1" Destination="W1"/>
  <Transition Source="W2" Event="E2" Destination="P2"/>
  <Transition Source="P2" Event="Beta2" Destination="B2"/>
  <Transition Source="P2" Event="Gama2" Destination="W2"/>
  <Transition Source="B2" Event="a2" Destination="W2"/>
  <Transition Source="WRe" Event="E2" Destination="R1"/>
  <Transition Source="WRe" Event="E3" Destination="R2"/>
  <Transition Source="R1" Event="Mi1" Action="Eta1M1" Destination="WRe"/>
  <Transition Source="R2" Event="Mi2" Action="Eta1M2" Destination="WRe"/>
</Transitions>
</PerformCharts>

```

FIGURE 4.5 - (conclusion)

CHAPTER 5

On dealing with PcML for Performance Evaluation

The software for PerformCharts was developed in C++. As mentioned previously, in order to compute the performance measures of a given reactive system, it is necessary to code first the specification of the system itself as a block within PerformCharts' main program in C++. In the same main program the second part consists of generating a Markov chain and finally the third part deals with invoking the methods to determine the performance measures. 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. The complete main program for the system represented in Figure 4.3 is shown in Figure 5.1.

```

#include <iostream.h>
#include <fstream.h>

#include "stcht.h"
#include "graphbas.h"
#include "graphgen.h"
#include "mkchain.h"

void main ()
{

    ofstream fout ("c:\\Statecharts\\Stcharts2\\out\\petri1.txt" );

    Statechart System;

// States
    System.createRoot("System",AND);
    System.createSonState ("Machine 1",OR,"System");
    System.createSonState ("P1",BASIC,"Machine 1");
    System.createSonState ("W1",BASIC,"Machine 1");
    System.createSonState ("B1",BASIC,"Machine 1");
    System.createSonState ("Machine 2",OR,"System");
    System.createSonState ("P2",BASIC,"Machine 2");
    System.createSonState ("W2",BASIC,"Machine 2");
    System.createSonState ("B2",BASIC,"Machine 2");
    System.createSonState ("Repairer",OR,"System");
    System.createSonState ("WRe",BASIC,"Repairer");
    System.createSonState ("R1",BASIC,"Repairer");
    System.createSonState ("R2",BASIC,"Repairer");
    System.setDefaultEntry ("Machine 1","P1");
    System.setDefaultEntry ("Machine 2","W2");
    System.setDefaultEntry ("Repairer","WRe");

// Defining Primitive Events right now for all the States. This will avoid
// consistency error because when an action is declared that event
// must have been defined earlier.

// Primitive Events (Stochastic Events and Actions)
    System.createPrimEvent ("Beta1",0.1);
    System.createPrimEvent ("Gama1",5.0);
    System.createPrimEvent ("Beta2",0.2);
    System.createPrimEvent ("Gama2",5.0);
    System.createPrimEvent ("Mi1",3.0);
    System.createPrimEvent ("Mi2",3.0);
    System.createPrimEvent("a1");
    System.createPrimEvent("a2");

```

FIGURE 5.1 – Main program in C++ for the System represented in Figure 4.3 (continue)

```

// Conditions
InStateCondition *inC1 = System.createInStateCondition("W2");
InStateCondition *inC2 = System.createInStateCondition("B1");
InStateCondition *inC3 = System.createInStateCondition("B2");
NotCondition *ncC4 = System.createNotCondition(*inC2);
ComposedCondition *ccC5 = System.createComposedCondition(ANDCOND,*inC3,*ncC4);

// Actions
EventTriggeringAction *Eta1M1 = System.createEventTrigAction("a1");
EventTriggeringAction *Eta1M2 = System.createEventTrigAction("a2");

//Events
TrueCondEvent *tevE1 = System.createTrueCondEvent (*inC1);
TrueCondEvent *tevE2 = System.createTrueCondEvent (*inC2);
TrueCondEvent *tevE3 = System.createTrueCondEvent (*ccC5);

//Probabilities

// Transitions
Transition *transW1_P1= System.createTransition (*tevE1);
System.addSourceNode (*transW1_P1, "W1" );
System.addDestinationNode (*transW1_P1, "P1");

Transition *transP1_B1= System.createTransition ("Beta1");
System.addSourceNode (*transP1_B1, "P1" );
System.addDestinationNode (*transP1_B1, "B1");

Transition *transP1_W1= System.createTransition ("Gama1");
System.addSourceNode (*transP1_W1, "P1" );
System.addDestinationNode (*transP1_W1, "W1");

Transition *transB1_W1= System.createTransition ("a1");
System.addSourceNode (*transB1_W1, "B1" );
System.addDestinationNode (*transB1_W1, "W1");

Transition *transW2_P2= System.createTransition (*tevE2);
System.addSourceNode (*transW2_P2, "W2" );
System.addDestinationNode (*transW2_P2, "P2");

Transition *transP2_B2= System.createTransition ("Beta2");
System.addSourceNode (*transP2_B2, "P2" );
System.addDestinationNode (*transP2_B2, "B2");

Transition *transP2_W2= System.createTransition ("Gama2");
System.addSourceNode (*transP2_W2, "P2" );
System.addDestinationNode (*transP2_W2, "W2");

Transition *transB2_W2= System.createTransition ("a2");
System.addSourceNode (*transB2_W2, "B2" );
System.addDestinationNode (*transB2_W2, "W2");

```

FIGURE 5.1 - (continuation)
(continue)

```

Transition *transWRe_R1= System.createTransition (*tevE2);
System.addSourceNode (*transWRe_R1, "WRe" );
System.addDestinationNode (*transWRe_R1, "R1");

Transition *transWRe_R2= System.createTransition (*tevE3);
System.addSourceNode (*transWRe_R2, "WRe" );
System.addDestinationNode (*transWRe_R2, "R2");

Transition *transR1_WRe= System.createTransition ("Mi1", *Eta1M1);
System.addSourceNode (*transR1_WRe, "R1" );
System.addDestinationNode (*transR1_WRe, "WRe");

Transition *transR2_WRe= System.createTransition ("Mi2", *Eta1M2);
System.addSourceNode (*transR2_WRe, "R2" );
System.addDestinationNode (*transR2_WRe, "WRe");

System.printStates( fout );
System.printTransitions ( fout );

flush ( fout );
GraphBase gb;
GraphGenerator gen;

//Perform reaction - generate all possible Configurations

gen.generateGraph (System, gb );

gb.printOn ( fout );

flush ( fout );

//Determine steady-state probabilities

MarkovChainGenerator mk ( gb );

mk.generateLimitingProbabilities ( );

mk.printLimitProbabilities ( fout );

mk.printCompStLimitProb ( fout );

}

```

FIGURE 5.1 - (conclusion)

The main idea here is to take the PcML and convert it into the main program with which one can link to other classes in order to generate the desired performance measures. As already mentioned earlier, two approaches have been used in dealing with PcML: Perl and Java. It is worth stressing that both these languages can deal with XML tags and therefore these resources have been used.

Schema PcMLschema.xsd
Description of the PcML Elements


Elements	Simple types
Actions	floatProbType
ANDCond	floatStateType
Author	rootType
ComposedCondition	stateType
Conditions	
Email	
EnteredState	
Events	
EventTriggerAction	
ExitedState	
FalseCondition	
History	
HStar	
Info	
InState	
Name	
NotCondition	
ORCond	
PcML	
Probabilities	
Probability	
Root	
State	
States	
Stochastic	
Transition	
Transitions	
TrueCondition	

element Actions

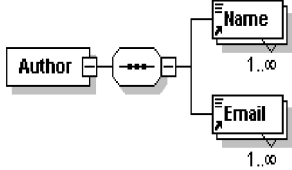
diagram	<pre> classDiagram class Actions class EventTriggerAction Actions "1" -- "*" EventTriggerAction </pre>
children	EventTriggerAction
used by	PcML element
source	<pre> <xs:element name="Actions"> <xs:complexType> <xs:sequence> <xs:element ref="EventTriggerAction" maxOccurs="unbounded"/> </xs:sequence> </xs:complexType> </xs:element> </pre>
Use in PcML	<pre> <Actions> <EventTriggerAction Name=""/> <EventTriggerAction Name=""/> </pre>

</Actions>

element ANDCond

diagram			
used by	ComposedCondition		element
attributes	Name Type Use Default Fixed Annotation	Cond1 xs:string required	Cond2 xs:string required
source	<pre><xs:element name="ANDCond"> <xs:complexType> <xs:attribute name="Cond1" type="xs:string" use="required"/> <xs:attribute name="Cond2" type="xs:string" use="required"/> </xs:complexType> </xs:element></pre>		
PcML use	<pre><ANDCond Cond1="" Cond2=""/></pre>		

element Author

diagram			
children	Name Email		
used by	Info		element
source	<pre><xs:element name="Author"> <xs:complexType> <xs:sequence> <xs:element ref="Name" maxOccurs="unbounded"/> <xs:element ref="Email" maxOccurs="unbounded"/> </xs:sequence> </xs:complexType> </xs:element></pre>		
Use in PcML:	<pre><Author> <Name/> <Email/> </Author></pre>		

<Email/>

 </Author>

element ComposedCondition


diagram		
children	ANDCond ORCond	
used by	Conditions	element
attributes	Name Type Use Default Fixed Annotation	Name xs:string required
source	<pre> <xs:element name="ComposedCondition"> <xs:complexType> <xs:choice> <xs:element ref="ANDCond"/> <xs:element ref="ORCond"/> </xs:choice> <xs:attribute name="Name" type="xs:string" use="required"/> </xs:complexType> </xs:element> </pre>	
PcML use	<pre> <ComposedCondition Name=""> <ANDCond Cond1="" Cond2=""/> </ComposedCondition> <ComposedCondition Name=""> <ORCond Cond1="" Cond2=""/> </ComposedCondition> </pre>	

element Conditions


diagram		
children	InState NotCondition ComposedCondition	
used by	PcML	element
source	<pre> <xs:element name="Conditions"> <xs:complexType> <xs:choice maxOccurs="unbounded"> <xs:element ref="InState"/> <xs:element ref="NotCondition"/> </xs:choice> </xs:complexType> </xs:element> </pre>	

	<pre><xs:element ref="ComposedCondition"/> </xs:choice> </xs:complexType> </xs:element></pre>
<p>Use in PcML</p> <p><code><Conditions> ... </Conditions></code></p> <p>The Primitive Conditions set defines, basically, the boolean values true and false to be evaluated during a transition by the system. In addition, composed conditions also can be used in the specification of a system. The condition types implemented in the PerformCharts are:</p> <p><code><InState Name="" State=""/></code> True if a component of a system is in a state specified as parameter.</p> <p><code><NotCondition Name="" Condition=""/></code> The opposite value of a condition specified as parameter.</p> <p><code><ComposedCondition Name="">..... </ComposedCondition></code> Composed by more than one condition by the logical operators And and OR.</p>	

element Email

diagram	
type	restriction of xs:string
used by	Author element
facets	<code>[p{L}_-]+\(\.[p{L}_-]+\)*@[p{L}_]+\(\.[p{L}_-]+\)+</code> pattern
source	<pre><xs:element name="Email"> <xs:simpleType> <xs:restriction base="xs:string"> <xs:pattern value="[p{L}_-]+\(\.[p{L}_-]+\)*@[p{L}_]+\(\.[p{L}_-]+\)+"/> </xs:restriction> </xs:simpleType> </xs:element></pre>

element EnteredState

diagram																			
used by	Events element																		
attributes	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 33%;">Name</td> <td style="width: 33%;">Name</td> <td style="width: 33%;">State</td> </tr> <tr> <td>Type</td> <td>xs:string</td> <td>xs:string</td> </tr> <tr> <td>Use</td> <td>required</td> <td>required</td> </tr> <tr> <td>Default</td> <td></td> <td></td> </tr> <tr> <td>Fixed</td> <td></td> <td></td> </tr> <tr> <td>Annotation</td> <td></td> <td></td> </tr> </table>	Name	Name	State	Type	xs:string	xs:string	Use	required	required	Default			Fixed			Annotation		
Name	Name	State																	
Type	xs:string	xs:string																	
Use	required	required																	
Default																			
Fixed																			
Annotation																			
source	<pre><xs:element name="EnteredState"> <xs:complexType> <xs:attribute name="Name" type="xs:string" use="required"/> <xs:attribute name="State" type="xs:string" use="required"/> </xs:complexType> </xs:element></pre>																		

Use in PcML

`<EnteredState Name="" State=""/>`

element Events

diagram			
children	TrueCondition FalseCondition EnteredState ExitedState Stochastic		
used by	PcML	element	
source	<pre> <xs:element name="Events"> <xs:complexType> <xs:choice maxOccurs="unbounded"> <xs:element ref="TrueCondition"/> <xs:element ref="FalseCondition"/> <xs:element ref="EnteredState"/> <xs:element ref="ExitedState"/> <xs:element ref="Stochastic"/> </xs:choice> </xs:complexType> </xs:element> </pre>		

element EventTriggerAction


diagram			
used by	Actions	element	
attributes	Name Type Use Default Fixed Annotation	Name xs:string required	Event xs:string required
source	<pre> <xs:element name="EventTriggerAction"> <xs:complexType> <xs:attribute name="Name" type="xs:string" use="required"/> <xs:attribute name="Event" type="xs:string" use="required"/> </xs:complexType> </xs:element> </pre>		

element ExitedState


diagram			
---------	--	--	--

used by	Events element		
attributes	Name Type Use Default Fixed Annotation	Name xs:string required	State xs:string required
source	<pre><xs:element name="ExitedState"> <xs:complexType> <xs:attribute name="Name" type="xs:string" use="required"/> <xs:attribute name="State" type="xs:string" use="required"/> </xs:complexType> </xs:element></pre>		
Use in PcML			
<pre><ExitedState Name="" State=""/></pre>			

element **FalseCondition**

diagram			
used by	Events element		
attributes	Name Type Use Default Fixed Annotation	Name xs:string required	Condition xs:string required
source	<pre><xs:element name="FalseCondition"> <xs:complexType> <xs:attribute name="Name" type="xs:string" use="required"/> <xs:attribute name="Condition" type="xs:string" use="required"/> </xs:complexType> </xs:element></pre>		
Use in PcML			
<pre><FalseCondition Name="" Condition=""/></pre>			

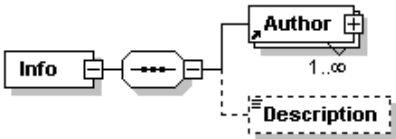
element **History**

diagram			
used by	State element		
source	<pre><xs:element name="History"> <xs:complexType/> </xs:element></pre>		


element HStar

diagram		
used by	State	element
source	<pre><xs:element name="HStar"> <xs:complexType/> </xs:element></pre>	

element Info

diagram		
children	Author Description	
used by	PcML	element
source	<pre><xs:element name="Info"> <xs:complexType> <xs:sequence> <xs:element ref="Author" maxOccurs="unbounded"/> <xs:element name="Description" type="xs:string" minOccurs="0"/> </xs:sequence> </xs:complexType> </xs:element></pre>	
Use in PcML:	<pre><Info> <Author> ... </Author> <Description> ... </Description> </Info></pre>	

element Info/Description

diagram		
type	xs:string	
source	<pre><xs:element name="Description" type="xs:string" minOccurs="0"/></pre>	
Use in PcML:	<pre><Description> ... </Description></pre>	

element InState


diagram		
used by	Conditions	element

attributes	Name Type Use Default Fixed Annotation	Name xs:string required	State xs:string required
source	<pre><xs:element name="InState"> <xs:complexType> <xs:attribute name="Name" type="xs:string" use="required"/> <xs:attribute name="State" type="xs:string" use="required"/> </xs:complexType> </xs:element></pre>		


Use in PcML

`<InState Name="" State=""/>`

element **Name**

diagram			
type	xs:string		
used by	Author		element
source	<code><xs:element name="Name" type="xs:string"/></code>		

element **NotCondition**

diagram			
used by	Conditions		element
attributes	Name Type Use Default Fixed Annotation	Name xs:string required	Condition xs:string required
source	<pre><xs:element name="NotCondition"> <xs:complexType> <xs:attribute name="Name" type="xs:string" use="required"/> <xs:attribute name="Condition" type="xs:string" use="required"/> </xs:complexType> </xs:element></pre>		

element **ORCond**

diagram			
---------	-------------------------------------------------------------------------------------	--	--

used by	ComposedCondition element		
attributes	Name Type Use Default Fixed Annotation	Cond1 xs:string required	Cond2 xs:string required
source	<pre> <xs:element name="ORCond"> <xs:complexType> <xs:attribute name="Cond1" type="xs:string" use="required"/> <xs:attribute name="Cond2" type="xs:string" use="required"/> </xs:complexType> </xs:element> </pre>		

element PcML

diagram			
children	Info States Conditions Actions Events Probabilities Transitions		
attributes	Name Type Use Default Fixed Annotation	Title xs:string required	Date xs:date
source	<pre> <xs:element name="PcML"> <xs:complexType> <xs:sequence> <xs:element ref="Info" minOccurs="0"/> <xs:element ref="States"/> <xs:element ref="Conditions" minOccurs="0"/> <xs:element ref="Actions" minOccurs="0"/> <xs:element ref="Events"/> <xs:element ref="Probabilities" minOccurs="0"/> <xs:element ref="Transitions"/> </xs:sequence> <xs:attribute name="Title" type="xs:string" use="required"/> <xs:attribute name="Date" type="xs:date"/> </xs:complexType> </xs:element> </pre>		

element Probabilities

diagram			
children	Probability		
used by	PcML		element
source	<pre><xs:element name="Probabilities"> <xs:complexType> <xs:sequence> <xs:element ref="Probability" maxOccurs="unbounded"/> </xs:sequence> </xs:complexType> </xs:element></pre>		
Use in PcML	<pre><Probabilities> <Probability Name="" Value=""/> </Probabilities></pre>		

element Probability

diagram			
used by	Probabilities		element
attributes	Name Type Use Default Fixed Annotation	Name xs:string required	Value floatProbType required
source	<pre><xs:element name="Probability"> <xs:complexType> <xs:attribute name="Name" type="xs:string" use="required"/> <xs:attribute name="Value" type="floatProbType" use="required"/> </xs:complexType> </xs:element></pre>		
Use in PcML	<pre><Probability Name="" Value=""/></pre>		

element Root


diagram			
children	State		
used by	States		element

attributes	Name Type Use Default Fixed Annotation	Name xs:string required	Type rootType required
source	<pre> <xs:element name="Root"> <xs:complexType> <xs:sequence> <xs:element ref="State" maxOccurs="unbounded"/> </xs:sequence> <xs:attribute name="Name" type="xs:string" use="required"/> <xs:attribute name="Type" type="rootType" use="required"/> </xs:complexType> </xs:element> </pre>		

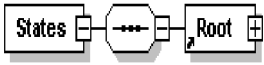
element State

diagram				
children	EMPTY State History HStar			
used by	Root State			elements
attributes	Name Type Use Default Fixed Annotation	Name xs:string required	Type stateType required	Default xs:string
source	<pre> <xs:element name="State"> <xs:complexType> <xs:sequence> <xs:choice minOccurs="0" maxOccurs="unbounded"> <xs:element name="EMPTY"> <xs:complexType/> </xs:element> <xs:element ref="State"/> </xs:choice> <xs:choice minOccurs="0"> <xs:element ref="History"/> <xs:element ref="HStar"/> </xs:choice> </xs:sequence> <xs:attribute name="Name" type="xs:string" use="required"/> <xs:attribute name="Type" type="stateType" use="required"/> <xs:attribute name="Default" type="xs:string"/> </xs:complexType> </xs:element> </pre>			


element State/EMPTY

diagram	
source	<pre><xs:element name="EMPTY"> <xs:complexType/> </xs:element></pre>


element States

diagram	
children	Root
used by	PcML element
source	<pre><xs:element name="States"> <xs:complexType> <xs:sequence> <xs:element ref="Root"/> </xs:sequence> </xs:complexType> </xs:element></pre>

element Stochastic

diagram																			
used by	Events element																		
attributes	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 33%;">Name</td> <td style="width: 33%;">Name</td> <td style="width: 33%;">Value</td> </tr> <tr> <td>Type</td> <td>xs:string</td> <td>floatStateType</td> </tr> <tr> <td>Use</td> <td>required</td> <td>required</td> </tr> <tr> <td>Default</td> <td></td> <td></td> </tr> <tr> <td>Fixed</td> <td></td> <td></td> </tr> <tr> <td>Annotation</td> <td></td> <td></td> </tr> </table>	Name	Name	Value	Type	xs:string	floatStateType	Use	required	required	Default			Fixed			Annotation		
Name	Name	Value																	
Type	xs:string	floatStateType																	
Use	required	required																	
Default																			
Fixed																			
Annotation																			
source	<pre><xs:element name="Stochastic"> <xs:complexType> <xs:attribute name="Name" type="xs:string" use="required"/> <xs:attribute name="Value" type="floatStateType" use="required"/> </xs:complexType> </xs:element></pre>																		

element Transition

diagram	
used by	Transitions element

attributes	Name Type Use Default Fixed Annotation	Source xs:string required	Event xs:string required	Condition xs:string	Probability xs:string	Action xs:string	Destination xs:string required
source	<pre> <xs:element name="Transition"> <xs:complexType> <xs:attribute name="Source" type="xs:string" use="required"/> <xs:attribute name="Event" type="xs:string" use="required"/> <xs:attribute name="Condition" type="xs:string"/> <xs:attribute name="Probability" type="xs:string"/> <xs:attribute name="Action" type="xs:string"/> <xs:attribute name="Destination" type="xs:string" use="required"/> </xs:complexType> </xs:element> </pre>						

element Transitions

diagram							
children	Transition						
used by	PcML						element
source	<pre> <xs:element name="Transitions"> <xs:complexType> <xs:sequence> <xs:element ref="Transition" maxOccurs="unbounded"/> </xs:sequence> </xs:complexType> </xs:element> </pre>						

element TrueCondition

diagram							
used by	Events						element
attributes	Name Type Use Default Fixed Annotation	Name xs:string required					Condition xs:string required
source	<pre> <xs:element name="TrueCondition"> <xs:complexType> <xs:attribute name="Name" type="xs:string" use="required"/> <xs:attribute name="Condition" type="xs:string" use="required"/> </xs:complexType> </xs:element> </pre>						

simpleType floatProbType

type	restriction of xs:float	
used by	Probability/@Value	attribute
facets		maxInclusive
nclusive	1.0	
	0.0	minExclusive
source	<pre><xs:simpleType name="floatProbType"> <xs:restriction base="xs:float"> <xs:maxInclusive value="1.0"/> <xs:minExclusive value="0.0"/> </xs:restriction> </xs:simpleType></pre>	

simpleType floatStateType

type	restriction of xs:float	
used by	Stochastic/@Value	attribute
facets	0.0	minExclusive
source	<pre><xs:simpleType name="floatStateType"> <xs:restriction base="xs:float"> <xs:minExclusive value="0.0"/> </xs:restriction> </xs:simpleType></pre>	

simpleType rootType

type	restriction of xs:string	
used by	Root/@Type	attribute
facets	AND	enumeration
	OR	enumeration
source	<pre><xs:simpleType name="rootType"> <xs:restriction base="xs:string"> <xs:enumeration value="AND"/> <xs:enumeration value="OR"/> </xs:restriction> </xs:simpleType></pre>	

simpleType	stateType
type	restriction of xs:string
used by	State/@Type attribute
facets	
enumeration	<p>BASIC enu</p> <p>XOR enumeration</p> <p>AND enumeration</p>
source	<pre> <xs:simpleType name="stateType"> <xs:restriction base="xs:string"> <xs:enumeration value="BASIC"/> <xs:enumeration value="XOR"/> <xs:enumeration value="AND"/> </xs:restriction> </xs:simpleType> </pre>

Both the languages, Perl and Java, can deal with XML tags and these resources have been used. Both the implementations access and manipulate the PcML document by means of DOM(Document Object Model)-based parsing. This DOM 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 (Icarus, 2002). 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. In Perl, it is implemented as a package XML::DOM. Thus, both programs read the PcML file and, according to the rules established in a DTD/Schema, check if it is well formed if the tags contained in it are valid. Errors are reported if they are found.

Once the PcML specification is converted into a main program, it has to be linked to the appropriate library to generate an executable file of the PerformCharts ready to be run. 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 (RAY, 2002; RIEHL e STERIN, 2002)

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 (MARUYAMA, 2002) and (VELOSO, 2003).

Now, some examples of PcML definitions (for the reactive system in Figure 4.3) and their corresponding C++ commands are in order.

By specifying the root and a State in PcML as

```
<Root Name="System" Type="AND">
```

```
<State Name="Machine1" Type="XOR" Default="P1">
```

the following C++ commands are generated:

```

Statechart System;
System.createRoot("System",AND);
System.createSonState ("Machine1",OR,"System");
System.setDefaultEntry ("Machine1","P1");

```

One can observe that the name assigned to the root state is also assigned to the system being modeled. The example shown above does not use any entry by History feature. In the case it were used, a command line similar to `System.createHistoryEntry("State")` would be generated.

Next, the condition specification in PcML:

```
<InState Name="C1" State="W2"/>
```

will generate:

```
InStateCondition *inC1 = System.createInStateCondition("W2");
```

Based on the appropriate definitions of the events, function calls have to be generated in the C++ program. 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 event 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. The following specification in PcML:

```
<Events>
```

```
...
```

```
<Stochastic Name="Mi1" Value="3.0"/>
```

```
...
```

```
</Events>
```

PcML block shows a typical specification of Action.

```
<Actions>
```

```
<EventTriggerAction Name="Eta1M1" Event="a1"/>
```

```
...
```

```
</Actions>
```

An example for True Condition Event is shown. Remember that the Condition C1 must

have been defined earlier:

```
<Events>
<TrueCondition Name="E1" Condition="C1"/>
...
</Events>
```

will generate the following C++ main program lines:

```
System.createPrimEvent ("Mi1",3.0);
System.createPrimEvent(a1");
EventTriggeringAction *Eta1M1 = System.createEventTrigAction("a1");
TrueCondEvent *tevE1 = 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 4.3 has no such events.

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

Here is one example of the creation of a transition, its specification and the C++ line command:

```
<Transition Source="W1" Event="E1" Destination="P1"/>
<ransition Source="R1" Event="Mi1" Action="EtaM1" Destination="Wre"/>
```

will generate the following lines in the main program:

```
Transition *transW1_P1 = System.createTransition (*tevE1);
System.addSourceNode (*transW1_P1,"W1");
System.addDestinationNode (*transW1_P1,"P1");
Transition *transR1_WRe= System.createTransition ("Mi1", *Eta1M1);
System.addSourceNode (*transR1_WRe, "R1" );
System.addDestinationNode (*transR1_WRe, "WRe" );
```

Finally, the following C++ program lines are the function calls that will generate the performance evaluation:

```
GraphBase gb;
GraphGenerator gen;
//Perform reaction - generate all possible Configurations
gen.generateGraph (System, gb );
gb.printOn ( fout );
flush ( fout );
//Determine steady-state probabilities
MarkovChainGenerator mk ( gb );
```



```
mk.generateLimitingProbabilities ( );  
mk.printLimitProbabilities ( fout );  
mk.printCompStLimitProb ( fout );
```

CHAPTER 6

Conclusions

6.1 Conclusions

Complex reactive systems are one of a kind where many intricacies have to be represented. Statecharts have powerful features where it is easier to represent such systems in most of the cases. In case of performance models, it was possible to associate Statecharts representation to a mathematical solution, in particular to Markov chains from which performance measures of a reactive system can be 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. Future 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 measurements of the given system. The main idea is to convert the graphical interface to PcML. However, other approaches of generating the performance measurements without having to convert into a main program are under consideration.

REFERENCES

10

INTERNATIONAL ORGANIZATION STANDARDIZATION, ISO 8879:1986, Nov 2002. Standard Generalized Markup Language (SGML)

<http://www.iso.ch/cate/d16387.html>.

Bates, C. XML in theory & practice. local: [S.I.]: John Wiley Computer, 2003.

10

Bonneau, S.; Williams, k.; Tennison, J. XML design handbook. local: [S.I.]: Wrox Press LTD., 2003. 10

Consortium, W. W. W. Extensible markup language (XML) 1.0. Available in <http://www.w3.org/XML/Activity>, Access in Sept 2002a. 14

———. Extensible markup language (XML) 1.0. Available in <http://www.w3.org/TR/NOTE-xml-schema-req>, Access in Nov 2002b. 14

———. Leading the web to its full potential... Available in <http://www.w3.org>, Access in Sept 2002c. 10

Drusinsky, D.; Harel, D. Using statecharts for hardware description and synthesis. In: IEEE Transactions on Computer-Aided Design, v. 8, n. 7, p. 798–807, 1989. 5

Graham, I. XML specification guide. local: [S.I.]: John Wiley Computer, 1999. 10

Harel, D. Statecharts: a visual formalism for complex systems. Science of Computer Programming, v. 8, p. 231–274, 1987. 5, 7

Harel, D.; Lachover, H.; Naamad, A.; Pnueli, A.; Politi, M.; Sherman, R.; Shtull-Trautin, A.; Trakhtenbrot, M. STATEMATE: A working environment for the development of complex reactive systems. In: IEEE Transactions on Software Engineering, v. 16, n. 4, p. 403–414, 1990. 5

Harel, D.; Naamad, A. The STATEMATE Semantics of Statecharts. ACM Transactions on Software Engineering, v. 5, n. 4, p. 293–333, 1996. 5, 7

Harel, D.; Pnueli, A.; Schmit, J.; Sherman, R. On the formal semantics of Statecharts, 1987. 5, 7

- Harel, D.; Politi, M. Modeling reactive systems with statecharts: the statemate approach. local: [S.I.]: McGraw-Hill, 1998. 5, 7
- Holzner, S. XML Complete. local: [S.I.]: McGraw-Hill, 1998. 10
- Icarus, M. Using perl with XML. Available in: http://www.devshed.com/Server_Side/Perl/PerlXML/PerlXML2/print_htm, Access in Sept 2002. DevShed.com. 39
- Maden, C. Creating documents in XML. local: [S.I.]: Oreilly & Associated, 2000. 10
- Maden, C.; Ray, E. Learning XML. local: [S.I.]: Oreilly & Associated, 2001. 10
- Maruyama, H. XML and Java - developing web application. local: [S.I.]: Addison Wesley, 2002. 39
- Philippe, B.; Saad, Y.; Stewart, W. Numerical methods in Markov chain modeling. Operations Research, v. 40, n. 6, p. 1156–1179, 1992. 5
- Pinnock, J.; Dix, C.; Rafter, J. Beginning XML. Wrox Press, 2001. 10
- Ray, E. T. Perl and XML. local: [S.I.]: Oreilly & Associates, Inc., 2002. 39
- Riehl, M.; Sterin, I. XML and Perl. local: [S.I.]: Macmillan Computer Pub, 2002. 39
- Silva, E.A.S.and Muntz, R. Computational methods to solve Markov Chains: applications to computing and communication systems, 1992. 5
- ST. Laurent, S. Building XML Applications. McGraw-Hill, USA, 1999. 10
- Veloso, R. R. Java e XML - guia de consulta rápida. local: Brazil: Novatec editora, 2003. 39
- Vijaykumar, N. Statecharts: Their use in specifying and dealing with Performance Models. São José dos Campos, Brazil. Tese – Aeronautical Institute of Technology (ITA), 1999. 5, 7
- Vijaykumar, N. L.; Carvalho, S.; Abdurahiman, V. On proposing statecharts to specify performance models. International Transactions in Operational Research, v. 9, n. 3, p. 321–336, 2002. 5, 7, 8
- Young, M. J. XML step by step. Local:USA: Microsoft Press., 2001. 10

APPENDIX A

PcML Schema

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- edited with XMLSPY v5 rel. 4 U (http://www.xmlspy.com) by Rener Hehe (RH) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0">
<xs:element name="PcML">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Info" minOccurs="0"/>
      <xs:element ref="States"/>
      <xs:element ref="Conditions" minOccurs="0"/>
      <xs:element ref="Actions" minOccurs="0"/>
      <xs:element ref="Events"/>
      <xs:element ref="Probabilities" minOccurs="0"/>
      <xs:element ref="Transitions"/>
    </xs:sequence>
    <xs:attribute name="Title" type="xs:string" use="required"/>
    <xs:attribute name="Date" type="xs:date"/>
  </xs:complexType>
</xs:element>
<xs:element name="Info">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Author" maxOccurs="unbounded"/>
      <xs:element name="Description" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Author">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Name" maxOccurs="unbounded"/>
      <xs:element ref="Email" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Name" type="xs:string"/>
<xs:element name="Email">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[\p{L}_-]+(\.[\p{L}_-]+)*@[\p{L}_-]+(\.[\p{L}_-]+)"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
<xs:element name="States">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="Root"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Root">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="State" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

```

        <xs:attribute name="Name" type="xs:string" use="required"/>
        <xs:attribute name="Type" type="rootType" use="required"/>
    </xs:complexType>
</xs:element>
<xs:element name="State">
    <xs:complexType>
        <xs:sequence>
            <xs:choice minOccurs="0" maxOccurs="unbounded">
                <xs:element name="EMPTY">
                    <xs:complexType/>
                </xs:element>
                <xs:element ref="State"/>
            </xs:choice>
            <xs:choice minOccurs="0">
                <xs:element ref="History"/>
                <xs:element ref="HStar"/>
            </xs:choice>
        </xs:sequence>
        <xs:attribute name="Name" type="xs:string" use="required"/>
        <xs:attribute name="Type" type="stateType" use="required"/>
        <xs:attribute name="Default" type="xs:string"/>
    </xs:complexType>
</xs:element>
<xs:element name="History">
    <xs:complexType/>
</xs:element>
<xs:element name="HStar">
    <xs:complexType/>
</xs:element>
<xs:element name="Conditions">
    <xs:complexType>
        <xs:choice maxOccurs="unbounded">
            <xs:element ref="InState"/>
            <xs:element ref="NotCondition"/>
            <xs:element ref="ComposedCondition"/>
        </xs:choice>
    </xs:complexType>
</xs:element>
<xs:element name="InState">
    <xs:complexType>
        <xs:attribute name="Name" type="xs:string" use="required"/>
        <xs:attribute name="State" type="xs:string" use="required"/>
    </xs:complexType>
</xs:element>
<xs:element name="NotCondition">
    <xs:complexType>
        <xs:attribute name="Name" type="xs:string" use="required"/>
        <xs:attribute name="Condition" type="xs:string" use="required"/>
    </xs:complexType>
</xs:element>
<xs:element name="ComposedCondition">
    <xs:complexType>
        <xs:choice>
            <xs:element ref="ANDCond"/>
            <xs:element ref="ORCond"/>
        </xs:choice>
    </xs:complexType>
</xs:element>

```



```

        <xs:attribute name="Name" type="xs:string" use="required"/>
    </xs:complexType>
</xs:element>
<xs:element name="ANDCond">
    <xs:complexType>
        <xs:attribute name="Cond1" type="xs:string" use="required"/>
        <xs:attribute name="Cond2" type="xs:string" use="required"/>
    </xs:complexType>
</xs:element>
<xs:element name="ORCond">
    <xs:complexType>
        <xs:attribute name="Cond1" type="xs:string" use="required"/>
        <xs:attribute name="Cond2" type="xs:string" use="required"/>
    </xs:complexType>
</xs:element>
<xs:element name="Actions">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="EventTriggerAction" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="EventTriggerAction">
    <xs:complexType>
        <xs:attribute name="Name" type="xs:string" use="required"/>
        <xs:attribute name="Event" type="xs:string" use="required"/>
    </xs:complexType>
</xs:element>
<xs:element name="Events">
    <xs:complexType>
        <xs:choice maxOccurs="unbounded">
            <xs:element ref="TrueCondition"/>
            <xs:element ref="FalseCondition"/>
            <xs:element ref="EnteredState"/>
            <xs:element ref="ExitedState"/>
            <xs:element ref="Stochastic"/>
        </xs:choice>
    </xs:complexType>
</xs:element>
<xs:element name="TrueCondition">
    <xs:complexType>
        <xs:attribute name="Name" type="xs:string" use="required"/>
        <xs:attribute name="Condition" type="xs:string" use="required"/>
    </xs:complexType>
</xs:element>
<xs:element name="FalseCondition">
    <xs:complexType>
        <xs:attribute name="Name" type="xs:string" use="required"/>
        <xs:attribute name="Condition" type="xs:string"
            use="required"/>
    </xs:complexType>
</xs:element>
<xs:element name="EnteredState">
    <xs:complexType>
        <xs:attribute name="Name" type="xs:string" use="required"/>

```

```

        <xs:attribute name="State" type="xs:string" use="required"/>
    </xs:complexType>
</xs:element>
<xs:element name="ExitedState">
    <xs:complexType>
        <xs:attribute name="Name" type="xs:string" use="required"/>
        <xs:attribute name="State" type="xs:string" use="required"/>
    </xs:complexType>
</xs:element>
<xs:element name="Stochastic">
    <xs:complexType>
        <xs:attribute name="Name" type="xs:string" use="required"/>
        <xs:attribute name="Value" type="floatStateType"
            use="required"/>
    </xs:complexType>
</xs:element>
<xs:element name="Probabilities">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="Probability" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="Probability">
    <xs:complexType>
        <xs:attribute name="Name" type="xs:string" use="required"/>
        <xs:attribute name="Value" type="floatProbType"
            use="required"/>
    </xs:complexType>
</xs:element>
<xs:element name="Transitions">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="Transition" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="Transition">
    <xs:complexType>
        <xs:attribute name="Source" type="xs:string" use="required"/>
        <xs:attribute name="Event" type="xs:string" use="required"/>
        <xs:attribute name="Condition" type="xs:string"/>
        <xs:attribute name="Probability" type="xs:string"/>
        <xs:attribute name="Action" type="xs:string"/>
        <xs:attribute name="Destination" type="xs:string" use="required"/>
    </xs:complexType>
</xs:element>
<xs:simpleType name="floatStateType">
    <xs:restriction base="xs:float">
        <xs:minExclusive value="0.0"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="floatProbType">
    <xs:restriction base="xs:float">
        <xs:maxInclusive value="1.0"/>
        <xs:minExclusive value="0.0"/>
    </xs:restriction>
</xs:simpleType>

```

```
        </xs:restriction>
</xs:simpleType>
<xs:simpleType name="stateType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="BASIC"/>
    <xs:enumeration value="XOR"/>
    <xs:enumeration value="AND"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name="rootType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="AND"/>
    <xs:enumeration value="OR"/>
  </xs:restriction>
</xs:simpleType>
</xs:schema>
```

INDEX

C++ Program, [20–22](#)

Conclusions, [43](#)

PcML, [5](#), [7](#), [9](#), [11](#), [19](#)

PCML Diagram, [11](#)

PCML schema, [52](#)

PcML Schema, [48](#)

PCML schema1, [48](#)

PCML schema2, [49](#)

PCML schema3, [50](#)

PCML schema4, [51](#)

PcML Specification, [16](#), [17](#)

PCML Structure, [13](#)

System, [14](#)

Transitions table, [15](#)

INDEX

C++ Program, [20–22](#)

Conclusions, [43](#)

PcML, [5](#), [7](#), [9](#), [11](#), [19](#)

PCML Diagram, [11](#)

PCML schema, [52](#)

PcML Schema, [48](#)

PCML schema1, [48](#)

PCML schema2, [49](#)

PCML schema3, [50](#)

PCML schema4, [51](#)

PcML Specification, [16](#), [17](#)

PCML Structure, [13](#)

System, [14](#)

Transitions table, [15](#)