



Ministério da  
Ciência e Tecnologia



INPE-15379-TDI/1398

## WEB-PERFORMCHARTS: A WEB-BASED TEST CASE GENERATOR FROM STATECHARTS MODELING

Alessandro Oliveira Arantes

Dissertação de Mestrado do Curso de Pós-Graduação em Computação Aplicada,  
orientada pelo Dr. Nandamudi Lankalapalli Vijaykumar, aprovada em 11 de agosto  
de 2008.

Registro do documento original:

<<http://urlib.net/sid.inpe.br/mtc-m18@80/2008/07.14.19.42>>

INPE  
São José dos Campos  
2008

## **PUBLICADO POR:**

Instituto Nacional de Pesquisas Espaciais - INPE

Gabinete do Diretor (GB)

Serviço de Informação e Documentação (SID)

Caixa Postal 515 - CEP 12.245-970

São José dos Campos - SP - Brasil

Tel.:(012) 3945-6911/6923

Fax: (012) 3945-6919

E-mail: [pubtc@sid.inpe.br](mailto:pubtc@sid.inpe.br)

## **CONSELHO DE EDITORAÇÃO:**

### **Presidente:**

Dr. Gerald Jean Francis Banon - Coordenação Observação da Terra (OBT)

### **Membros:**

Dr<sup>a</sup> Maria do Carmo de Andrade Nono - Conselho de Pós-Graduação

Dr. Haroldo Fraga de Campos Velho - Centro de Tecnologias Especiais (CTE)

Dr<sup>a</sup> Inez Staciarini Batista - Coordenação Ciências Espaciais e Atmosféricas (CEA)

Marciana Leite Ribeiro - Serviço de Informação e Documentação (SID)

Dr. Ralf Gielow - Centro de Previsão de Tempo e Estudos Climáticos (CPT)

Dr. Wilson Yamaguti - Coordenação Engenharia e Tecnologia Espacial (ETE)

## **BIBLIOTECA DIGITAL:**

Dr. Gerald Jean Francis Banon - Coordenação de Observação da Terra (OBT)

Marciana Leite Ribeiro - Serviço de Informação e Documentação (SID)

Jefferson Andrade Ancelmo - Serviço de Informação e Documentação (SID)

Simone A. Del-Ducca Barbedo - Serviço de Informação e Documentação (SID)

## **REVISÃO E NORMALIZAÇÃO DOCUMENTÁRIA:**

Marciana Leite Ribeiro - Serviço de Informação e Documentação (SID)

Marilúcia Santos Melo Cid - Serviço de Informação e Documentação (SID)

Yolanda Ribeiro da Silva Souza - Serviço de Informação e Documentação (SID)

## **EDITORAÇÃO ELETRÔNICA:**

Viveca Sant´Ana Lemos - Serviço de Informação e Documentação (SID)



Ministério da  
Ciência e Tecnologia



INPE-15379-TDI/1398

## WEB-PERFORMCHARTS: A WEB-BASED TEST CASE GENERATOR FROM STATECHARTS MODELING

Alessandro Oliveira Arantes

Dissertação de Mestrado do Curso de Pós-Graduação em Computação Aplicada,  
orientada pelo Dr. Nandamudi Lankalapalli Vijaykumar, aprovada em 11 de agosto  
de 2008.

Registro do documento original:

<<http://urlib.net/sid.inpe.br/mtc-m18@80/2008/07.14.19.42>>

INPE  
São José dos Campos  
2008

A14w Arantes, Alessandro Oliveira.

WEB-PerformCharts: a web-based test case generator from statecharts modeling / Alessandro Oliveira Arantes. – São José dos Campos: INPE, 2008.

86p. ; (INPE-15379-TDI/1398)

Dissertação (Mestrado em Computação Aplicada) – Instituto Nacional de Pesquisas Espaciais, São José dos Campos, 2008.

1. Colaborativo. 2. Statecharts. 3. Verificação e validação. 4. Casos de testes. 5. Testes de software. I. Título.

CDU 004.05

---

Copyright © 2008 do MCT/INPE. Nenhuma parte desta publicação pode ser reproduzida, armazenada em um sistema de recuperação, ou transmitida sob qualquer forma ou por qualquer meio, eletrônico, mecânico, fotográfico, microfílmico, reprográfico ou outros, sem a permissão escrita da Editora, com exceção de qualquer material fornecido especificamente no propósito de ser entrado e executado num sistema computacional, para o uso exclusivo do leitor da obra.

Copyright © 2008 by MCT/INPE. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, microfilming, recording or otherwise, without written permission from the Publisher, with the exception of any material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use of the reader of the work.

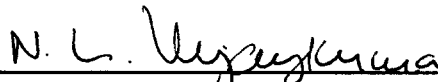
Aprovado (a) pela Banca Examinadora  
em cumprimento ao requisito exigido para  
obtenção do Título de Mestre em  
Computação Aplicada

Dr. Solon Venâncio de Carvalho



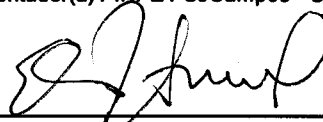
Presidente / INPE / SJC Campos - SP

Dr. Nandamudi Lankalapalli Vijaykumar



Orientador(a) / INPE / SJC Campos - SP

Dr. Edson Luiz Franca Senne



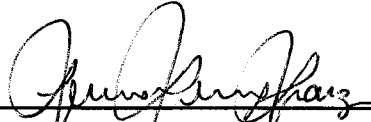
Membro da Banca / UNESP/GUARA / Guaratinguetá - SP

Dr. Fernando José de Oliveira Moreira



Convidado(a) / EMBRAER / SJC Campos - SP

Dra. Simone do Rocio Senger de Souza



Convidado(a) / USP-São Carlos / São Carlos - SP

Aluno (a): Alessandro Oliveira Arantes

São José dos Campos, 11 de agosto de 2008



*“Não sabendo que era impossível, ele foi lá e fez.”.*

*Lao-Tsé (filósofo chinês)*





*Aos colegas, docentes e discentes da Pós-Graduação do INPE.*



## **AGRADECIMENTOS**

Primeiramente, o maior dos agradecimentos ao Prof. Vijaykumar por ter disposto de seu tempo e dedicação para ser meu orientador e por todas as contribuições que foram fundamentais no desenrolar deste trabalho; assim como, também, estou muito grato ao Prof. Valdivino que acompanhou atenciosamente o processo e teve uma participação essencial para o direcionamento e andamento do trabalho.

Gostaria de agradecer também às colegas Ana Silvia e Danielle que muito ajudaram no desenvolver do trabalho, à Mônica que dedicou muito de sua atenção sendo minha orientadora no IEAv; e também à Maria de Fátima que me deu um apoio fundamental para o ingresso no curso.

Não posso deixar de agradecer a minha esposa Érica que não se deixou incomodar (muito) com as minhas noites de trabalho em casa; e também meus pais, familiares e amigos que sempre torceram incondicionalmente por mim.

Também agradeço a Deus por ter me dado saúde e disposição para que eu pudesse atingir esse importante marco em minha vida.

Da mesma forma, fico muito grato aos colegas do IEAv (Charbel, Afonso, Walter Jr. e Remo) que permitiram minha dedicação ao curso e me auxiliaram muito em minhas tarefas diárias.

E por último, mas não menos importante, eu ficarei eternamente grato a todos aqueles que de alguma forma contribuíram com o trabalho, ou mesmo aos que somente contribuíram para que minha passagem pelo curso fosse inesquecível.



# **WEB-PerformCharts: GERANDO CASOS DE TESTES VIA WEB A PARTIR DE ESPECIFICAÇÕES EM STATECHARTS**

## **RESUMO**

O desenvolvimento distribuído de software é uma realidade cada vez mais comum onde equipes espalhadas por um país ou mesmo pelo mundo podem trabalhar juntas no desenvolvimento de um produto. Nesse sentido, a utilização da internet é o recurso o qual possibilita o trabalho cooperativo entre profissionais geograficamente distantes. A presente dissertação propõe uma ferramenta acessível pela internet, WEB-PerformCharts, que adapta as rotinas da ferramenta PerformCharts possibilitando a geração e armazenamento de casos de teste remotamente via internet pelos testadores de software. O funcionamento da ferramenta proposta se baseia na especificação de sistemas reativos utilizando a técnica Statecharts e na geração de casos de teste para a mesma de acordo com alguns métodos disponíveis. A maior contribuição deste trabalho diz respeito ao estudo de métodos apropriados para a geração de casos de teste aplicados a software embarcado, além de propiciar a utilização da WEB-PerformCharts remotamente com o objetivo de dar suporte aos processos de testes em um ambiente de desenvolvimento distribuído.



## **ABSTRACT**

Distributed development of software is an increasing approach where teams spread over a country or even over the world can work together in order to develop the product. Web appears as a valuable resource enabling the cooperative development of software by professionals geographically distant from each other. This dissertation proposes a web-based tool, WEB-PerformCharts, which implements PerformCharts tool by adapting it to enable test designers to achieve generation of test sequences remotely via Internet. The goal of this proposed tool is to specify a reactive system in Statecharts, using the Web, and to generate test sequences according to a test case generation method. The main contribution of this dissertation is to investigate a test case generation method appropriate for space software specifications, besides enabling the use of WEB-PerformCharts through remote access with the objective of supporting the test process in a distributed development environment.





# SUMMARY

Page

**LIST OF FIGURES**  
**LIST OF TABLES**  
**LIST OF ABBREVIATIONS**

<b>1 INTRODUCTION</b> .....	<b>23</b>
1.1 Objective.....	26
<b>2 RELATED WORK</b> .....	<b>29</b>
2.1 Tests for Verification and Validation Activities .....	29
2.2 Software Modeling .....	32
2.3 FSM and Statecharts .....	33
2.4 PerformCharts .....	35
2.5 Testing Critical Systems within a Collaborative Scenario .....	37
<b>3 WEB-PERFORMCHARTS</b> .....	<b>41</b>
3.1 Architecture.....	41
3.2 Methods for Test Case Generation .....	43
3.2.1 Transition Tour (T-Method) .....	44
3.2.2 Switch Cover.....	46
3.3 Methodology used in WEB-PerformCharts .....	50
<b>4 RESULTS</b> .....	<b>57</b>
4.1 First Case Study .....	57
4.2 Second Case Study .....	58
4.3 Third Case Study .....	59
4.4 Fourth Case Study.....	63
4.5 Fifth Case Study .....	67
4.6 Sixth Case Study .....	70
4.7 Seventh Case Study .....	73
<b>5 CONCLUSION</b> .....	<b>79</b>
<b>REFERENCES</b> .....	<b>83</b>



## LIST OF FIGURES

1.1 - Fault, error and failure differences .....	24
2.1 - Statecharts representation of equipment with a repairer .....	36
2.2 - FSM of the example in Figure 2.1 .....	37
2.3 - Example of cooperative work. ....	40
3.1 - WEB-PerformCharts architecture .....	44
3.2 - Transition Tour algorithm .....	45
3.3 - Example of a simple FSM .....	46
3.4 - Demonstration of Switch Cover method. ....	47
3.5 - Graph eulerized by Condata tool.....	48
3.6 - Switch Cover algorithm .....	49
3.7 - PcML specification of modeling in Figure 2.1 .....	50
3.8 - Upload interface in web server .....	51
3.9 - FSM specified in XML .....	52
3.10 - Base of Facts for Condata tool.....	55
4.1 - Statecharts representation of TCP protocol behavior.....	59
4.2 - FSM generated from Figure 4.1 .....	60
4.3 - Statecharts representation of Producer-Consumer problem .....	63
4.4 - FSM generated from Figure 4.3 .....	64
4.5 - Statecharts representation of APEX system .....	69
4.6 - FSM generated from Figure 4.5. ....	69
4.7 - Statecharts representation of OBDH system.....	72
4.8 - Statecharts representation of APEX system detailed.....	74



## LIST OF TABLES

3.1 - Test Cases generated by Transition Tour method .....	53
3.2 - Test Cases generated by Switch Cover method .....	54
4.1 - Results obtained in first case study .....	58
4.2 - Results obtained in second case study .....	58
4.3 - Results obtained in third case study .....	60
4.4 - Test cases from Transition Tour in third case study .....	61
4.5 - Test cases from Switch Cover in third case study .....	62
4.6 - Results obtained in fourth case study .....	64
4.7 - Test cases from Transition Tour in fourth case study .....	65
4.8 - Test cases from Switch Cover in fourth case study .....	66
4.9 - Results obtained in fifth case study .....	68
4.10 - Results obtained in sixth case study .....	71
4.11 - Results obtained in seventh case study .....	73
4.12 - Test cases from Transition Tour in fifth case study .....	76
4.13 - Test cases from Switch Cover in fifth case study .....	78



## LIST OF ABBREVIATIONS

AGEDIS – Automated Generation and Execution of test suites for Distributed component-based Software  
APEX – Astrophysical Experiment  
ASML – Abstract State Machine Language  
CTA – Comando-Geral de Tecnologia Aeroespacial  
DFD – Data Flow Diagram  
ERD – Entity-Relationship diagram  
FSM – Finite State Machine  
HTML – HyperText Markup Language  
INPE – National Institute for Space Research  
IUT – Implementation Under Test  
OBDAH – On-Board Data Handler  
PCML – PerformCharts Markup Language  
PHP – Hypertext PreProcessor  
SDL – System Design Languages  
SQL – Structured Query Language  
STD – State-transition Diagram  
UML – Unified Modeling Language  
UNICAMP – State University of Campinas  
V&V – Verification and Validation  
XML – eXtensible Markup Language  
XSLT – Extensible Stylesheet Language Transformations





## 1 INTRODUCTION

Nowadays, it turns out to be more difficult to measure the importance of the software that surrounds the population. At the same time, it is quite impossible to imagine how complicated life would be without software. Software is embedded in computers, cell phones, production systems and many electronic equipment that employs some kind of control. There is no way to disagree that software is indeed in every aspect of modern life.

With the availability of modern technologies, it has become common for people to entrust several responsibilities in embedded software and they are also aware of the importance of its reliability as software is more and more used in independent decisions within important processes (SOMMERVILLE, 2003). However, in order to make the software more reliable, all functional and non-functional requirements to operate the software in the most varied platforms and under several conditions must be taken into consideration. This is the reason why software companies assign a significant amount of their budget to activities related to software tests or at least they should.

Historically in computing, tasks of software tests were left to a minor level plan and moreover it was not used to be considered as an attractive task being known as a tedious activity. However, this perception has been changing quickly among computing professionals and nowadays testing plays an extremely important role within the software's life cycle. Therefore, software developers must not save efforts to achieve a product with a maximum failure-proof feature. This role becomes even more critical when considering applications such as avionics and space as they involve significant amount of resources. Added to this fact are the eventual risks to human life and threats to environment.

Many professionals understand and use the terms fault, failure or error interchangeably; but in software engineering there are many references that distinguish these terms as in (BINDER, 2000) or (SOMMERVILLE, 2003) with

different definitions. Within the context of this dissertation the following definitions illustrated in Figure 1.1 will be used:

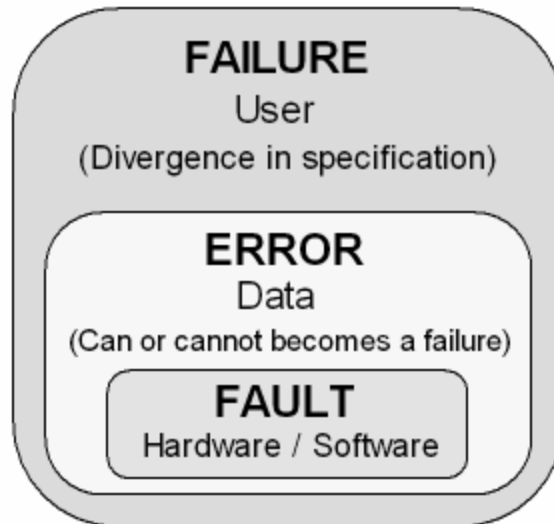


Figure 1.1 - Fault, error and failure differences

When a fault exists in hardware or software, it can be exemplified as a defective memory or source code mistakes. An error is the consequence of the fault within the program data, and it can (or cannot) become a failure sensed by user, when it is noticed that software does not match with its specification (BINDER, 2000).

Fault occurrence in several systems controlled by software causes inconvenience, but not serious damages. However, in certain systems software faults may result in significant economic losses, physical damages or threats to human life. These systems, in general, are known as critical systems (SOMMERVILLE, 2003).

Critical systems may be classified into three main categories (SOMMERVILLE, 2003):

- I. Security Critical System: A fault which can result in human threats, death or severe environmental damage. As an example one can cite a system that controls chemical plants.

- II. Mission Critical System: Faults may result in problems to achieve goals for some activity. Examples include navigation systems.
- III. Business Critical System: Faults may result in failure to finalize a trade deal. Good examples are systems that receive payments from bank customers.

Space software systems are classified as critical systems because their engineering processes demand high level and high cost technologies to perform complex tasks. Therefore, space agencies naturally demand higher software quality in lieu of huge investments for missions with scientific satellites (MATTIELLO et al., 2006).

For space research objectives, embedded software test planning can absolutely make the difference between success and failure, since entire mission depends on it. Software development for space applications is both a restricted area and difficult to work as it deals with highly specialized hardware with physical space limitations. For such critical systems, developers can not plan tests adopting a low cost and fast ad-hoc solutions; it must be formalized and planned as it is usually done during the analysis or development phases. Software testing must be conducted as a science and must be a formal process within the software development life cycle by assigning human and financial resources for this activity.

Space research organizations as INPE develops software for on-board computers embedded into satellites or stratospheric balloons that have a close interaction with the computer hardware, sensors, actuators and other devices. These on-board computers respond to stimuli, or events, from their devices making the embedded software a reactive system. Since these space missions are unmanned, the software is hard to be replaced in case of faults. Due to this fact, verification and validation phase during the entire software development life cycle is an important activity (SANTIAGO et al., 2006). INPE demands high software quality as huge investments in missions with scientific satellites have

become a routine and, many a time, teams involved in these missions are not exactly in one place as agencies exchange people with other national or international research agencies. In this scenario, an on-line collaborative tool is important to aid the software testing activities.

Collaborative work joins efforts of several members of a team to coordinate their tasks with an objective of reaching a specific goal. In particular, for software, collaborative web applications are powerful resources that can help different teams to cooperatively address process activities related to the software development life cycle, especially those related to testing.

### **1.1 Objectives**

In order to reduce costs, many software companies around the world are using computer-supported cooperative tools to overcome the geographical distances. Software development in geographically distributed settings is a natural trend in present days since internet is a powerful and a handy resource. Besides, web-based tools can aid different teams to cooperatively address process activities related to the software development life cycle (ARANTES et al., 2008). When dealing with software development, in particular with testing activities, software modeling is an important issue. Usually, formal methods are employed to represent a software specification in order to deal with them computationally for several purposes, such as automatic code generation, automatic test sequence generation, etc. In the context of this dissertation, the modeling technique employed to represent a software specification is Statecharts (HAREL, 1987) with an objective of automatically generate test cases. However, several methods to generate test cases have been developed as long as the specification is provided as an FSM. Therefore, while dealing with specification techniques such as Statecharts, they have to be converted into Finite State Machines (FSM). PerformCharts tool (VIJAYKUMAR et al., 2002) was developed to conduct this activity of converting a Statecharts representation into FSM. However, it is a standalone application. Moreover, its use is very limited for test sequence generation as it is integrated with yet another tool Condata

(MARTINS et al., 2000) that generates test cases and this is a very tedious and a time consuming process.

Therefore, as a primary objective, this work adapts PerformCharts to be implemented in a prototype tool, WEB-PerformCharts, for remote access in order to enable testing management capabilities through the web.

WEB-PerformCharts is able to read Statecharts specifications and generate proper test sequences according to a selected method. Another objective was to include the implementation of a method within WEB-PerformCharts in order to generate test sequences running independently without depending on another tool. For this purpose two methods were chosen to be implemented: T-Method (Transition Tour) (SIDHU and LEUNG, 1989) as it is a popular and easy to implement method; and Switch Cover (PIMONT and RAULT, 1979), as it was already being used through Condata.



## **2 RELATED WORK**

### **2.1 Tests for Verification and Validation Activities**

Software Verification and Validation (V&V) (PRESSMAN, 2000) process helps to ensure that software has all needed requirements in order to perform desired tasks (validation), and to ensure that all requirements are satisfied (verification). V & V is a systematic and technical evaluation where reviews and tests are done at the end of each development phase in order to ensure that requirements are correct and obeyed. Design, code, documentation and data must satisfy those requirements. The major V&V activities are reviews, walkthroughs, and testing.

Reviews are conducted during the end of each phase of the life cycle to determine whether specifications have been met. It is most effective when conducted by personnel not directly involved in the development. Walkthrough is a more detailed examination based on the source code debugging with the purpose of detecting errors. The group responsible for this is composed from development, test, and quality assurance teams.

Testing activity is the subject discussed in this dissertation, and it is an important phase in a V&V process because it is the software's operation with real or simulated inputs from real situations to demonstrate that software satisfies its requirements or, if it doesn't, to identify the differences between the expected outputs and obtained results. Tests can be applied in different phases of a system development process and there are several techniques available to plan and evaluate test cases (MYERS, 2004). In fact they complement each other and they can be classified as:

- I. Functional Tests: known as "black box tests", they are based on the software's specification but without any knowledge on its internal structure;

- II. Structural Tests: also known as “white box tests”, they are based on the internal structure of a given implementation;
- III. Error-Based Tests: introduction of common or typical errors into the software under test during the development process. Many references consider this technique from the same group of structural tests.

Generally structural tests are applied during the initial phases such as unit (function, method or class) tests, while functional tests are applied during integration and system testing.

Testing is specially important when dealing with complex and critical software such as space applications. In order to minimize the costs and the impact on the overall development process until the final product is released, modeling software behavior turns into an important technique that allows fixing errors in earlier phases (MYERS, 2004). A huge quantity of test sets must be applied to validate the product, which leads to the necessity of generating proper test sequences. Consequently the generation must rely entirely on a scientific basis in order to avoid their (test sequences) inadequacy in revealing errors.

Models can be used to describe behavior of a system in order to provide more resources for teams that deal with test development. Their use can be applied during several phases of software's life cycle such as specifications, code generation, reliability analysis and test case generation. Based on a model, the system behavior can be understood and the issue of test generation can be addressed (APFELBAUM and DOYLE, 1997). Test cases can be described generically as a sequence of actions that show a proper system behavior; thus, as the focus is in reactive systems, a test sequence is defined as an entire sequence of events that provide stimulus to the system, and the challenge is how to generate appropriate test sequences.

A significant number of methods to generate test sequences once the software is somehow represented has been published in the literature. The AGEDIS Consortium is an European agreement funded for joint cooperation research



programs related to software testing (HARTMAN, 2002). The objective of these research programs is to increase efficiency and quality in software industry by reducing costs in the testing phase through automating testing activities and this consequently leads to developing methodologies in order to guarantee software quality and reliability.

Also, the consortium investigates tools already developed since there are several commercial, academic or proprietary tools around the world developed based on some methodology and with an automatic test generation purpose. The CONFORMIQ TEST GENERATOR, REACTIS and TAUTTCN SUITE, are commercial tools which implement respectively UML (OMG, 2005), StateFlow (MATHWORKS, 2008) and SDL (ELLSBERGER et al, 1997) models as an input to generate test cases. An example of proprietary tool is ASML that implements abstract state machines models that depend on Microsoft Visual Studio.net. Some examples of academic tools are SPECTEST and TOSTER which are also UML based model. All these tools use some state-machine based specification, such as FSM, to simulate the application under test. State machines represent a set of states and transition arcs among these states labeled by an event, i.e., states change to other states based on the execution of these events (HARTMAN, 2002).

Test sequences from a state machine could be defined as a path from a given state (or configuration when parallel activities are considered) to another state or configuration that is reachable. If the software were modeled as a FSM, for example, some of the methods that can be applied are: T-Method, UIO-Method, D-Method and Switch Cover (LEE and YANNAKAKIS, 1996), (MARTINS et al., 2000), (MYERS, 2004) and (PIMONT and RAULT, 1979). However, some features usually present in complex software, like parallel activities and encapsulation, are very hard to model in FSM. Then, a better option when dealing with such kind of software is to use a higher-level modeling such as Statecharts (HAREL, 1987) and as Statecharts are formal, one can develop a computational algorithm to convert the specification into a FSM.

A proper way to fix possible faults and guarantee software correction is with exhaustive testing. The obvious and most natural choice one can think of is to test software by executing all possible inputs. This approach works fine for simple systems, but does not for huge systems with an intractable range of input domain (MYERS, 2004). If testing all input domains is intractable, one must use an alternative to select subsets of test data to be utilized for input. It is of fundamental importance that these testing techniques must be planned and conducted in a formal approach.

## **2.2 Software Modeling**

Any graphical or textual language that can be used to represent structured information or knowledge following a set of consistent rules is known as a modeling language. Modeling techniques have become common for domain-specific applications recently, in particular for software development (XIAO et al., 2007).

Several modeling techniques have been used to represent complex software in engineering systems. Each of them has the objective in satisfying the necessities of a specific domain. UML, for example, is a general-purpose modeling language that creates an abstract model of a given system (OMG, 2005); DFD uses relationships among processes of a system (GANE and SARSON, 1979); and, ERD describes a system by organizing its data as a primary objective (CHEN, 1976). Since these techniques are not quite formal, they are very hard to be computationally handled. However, there are also formal techniques that can be computationally handled representing a reactive system as a set of states and transitions among these states triggered by some stimulation. Such techniques that can be mentioned are Petri Nets (PETERSON, 1981), SDL (ELLSBERGER et al., 1997), Statecharts (HAREL, 1987) and others.

There is no unique method that can satisfy most of the modeling issues. Each of these has their advantages and drawbacks for a particular paradigm. Test

designers, following their human nature, are usually biased by a method they dominate well and it is highly unlikely that they would opt for a different technique even when it seems to have better features to represent a complex system behavior. Organizations as CTA and INPE deal with space applications which means complex software is developed not only to control the space missions as well as it is embedded in scientific instruments or experiments that fly on-board the spacecrafts. Such software has to go through a thorough verification and validation activities based on careful and organized tests. In order to conduct such type of testing, software must be specified formally in order to automate several processes involved.

### **2.3 FSM and Statecharts**

Reactive Systems are systems that maintain an ongoing interaction with their environment by responding to the interactions through some sort of processing. These systems are event-driven, since they continuously react to external or internal stimuli also known as events (HAREL and NAAMAD, 1996); and in particular within the context of this work, they are considered complex. A natural choice for representing reactive systems is FSM as it can be represented graphically by a state-transition diagram. However, features as depth and parallelism (usually common in modern complex reactive systems) are not easily specified in a straightforward manner through FSM. So, a formal higher-level technique should be investigated. This work is based on Statecharts to specify reactive systems (HAREL and POLITI, 1998).

Statecharts are graphical-oriented based on state-transition diagrams extending them by including notions of hierarchy, orthogonality and interdependence. They can specify reactive systems (HAREL and POLITI, 1998) and they are formal (HAREL et al., 1987) and (HAREL and POLITI, 1998) enabling computational handling. In order to represent a reactive system in Statecharts, one must make use of the following elements: States, Events, Conditions, Actions, Variables, Expressions and Transitions.

States are clustered to represent depth. One can combine a set of states with common transitions into a super-state and state refinement is achieved by means of *XOR* and *AND* decompositions. The former is used whenever an encapsulation is required. When a super-state *OR* is active, one (and only one) of its sub-states is indeed active. The latter is used to represent concurrency. When a super-state *AND* is active, all of its sub-states are active at the same time. A “basic state” is when there are no further refinements. In Statecharts global state of a given model is referred to as a configuration, that is, the basic active states of each parallel component.

By definition, when modeling a given system, there must always be an initial state also known as default state, which is the entry point. Another way to enter a system is through its history, i.e. when a system (or a sub-system) becomes active, the state most recently visited is activated. Symbol *H* has to be specified in order to use history. It is also possible to use the history all the way down to the lowest level (*H\**) (HAREL, 1987).

Events are fundamental to change system behavior so that configurations move to other configurations. Events have been classified into two categories: internal and external (HAREL, 1987). External events have to be explicitly stimulated. Internal (or immediate) events are those that are sensed automatically (not explicitly stimulated) and are enabled so that transitions are fired immediately. Statecharts have such built-in events: *true(condition)*, *false(condition)*, *entered(State)*, *exit(State)*. The basic element action can refer to change of a variable, expression or even another event. The original notation along a transition arc is *event[condition]/action*. This is interpreted as: when an event is enabled and the associated condition is satisfied, only then the transition takes place by moving from one state to another. Once the transition is fired, action is performed by changing a value of a variable or an expression or event continuing the reaction moving from one state (in another parallel component) to another.

## 2.4 PerformCharts

PerformCharts is a tool used to generate test sequences from Statecharts specifications. It was initially designed and developed to be used to evaluate performance of reactive systems by associating them to Markov Chains (VIJAYKUMAR et al., 2002). In PerformCharts, for the purpose of performance evaluation, external events are considered as stochastic following an exponential distribution. Internal events are considered as immediate as the transition takes place in zero time. Actions, in PerformCharts, besides changing values of a variable or an expression, are considered as internal events that affect other orthogonal components. In order to obtain performance evaluation, PerformCharts tool converts the Statecharts model into a Markov chain. This tool was written in C++ language.

Markov chain, in fact, can be represented graphically by a state-transition diagram. Based on this fact, PerformCharts tool has also been in use to generate test sequences. Therefore, in this case, a Statecharts model of a software specification is converted into an FSM from which test sequences can be derived. In this case, external events are just inputs without any stochastic information.

A graphical interface for Statecharts modeling is a related work that is under development. So, the specification of a reactive system in Statecharts and generation of FSM (or Markov chain) have to be coded as calls to methods in C++ language as a main module. In order to avoid this tedious coding, an XML-based (W3C, 2002) language PerformCharts Markup Language (PcML) (AMARAL et al., 2004) has been developed. PcML code is edited by any text editor and parsed by a Perl script that converts it to the main program in C++. Thus, this main program is linked and compiled with other classes and when executed, the corresponding FSM is generated.

As an example, Figure 2.1 shows a Statecharts representation of a Manufacturing system with a repairer. It has three parallel components that

correspond to two machines ( $E1$  and  $E2$ ) and a supervisor ( $Supervisor$ ) to repair any eventual failure of the machines. In case both the machines fail  $E1$  has a priority to be repaired. This priority is described by the event  $tr[in(B2) \wedge \neg in(B1)]$  meaning that the conditions in state  $B2$  ( $in(B2)$  – machine  $E2$  is down) and not in state  $B1$  ( $\neg in(B1)$  – machine  $E1$  is not down) have to be satisfied in order to fire the transition to repair  $E2$ . More details about the events and conditions in Statecharts can be seen in (HAREL, 1987). The list of stochastic events include  $a1, r1, f1, s1, a2, r2, f2$  and  $s2$ . Internal events are  $tr[in(B1)], tr[in(B2) \wedge \neg in(B1)]$ . Actions  $c1$  and  $c2$ , also considered as internal events, are triggered after the events  $s1$  and  $s2$  are executed. For example, once  $s1$  is triggered, a transition from state  $C1$  to  $WS$  (within the  $Supervisor$  component) is fired and this is followed by another transition associated to action  $c1$  moving from state  $B1$  to state  $W1$  (within  $E1$  component). The corresponding FSM from this example (after converted by PerformCharts tool) is shown in Figure 2.2.

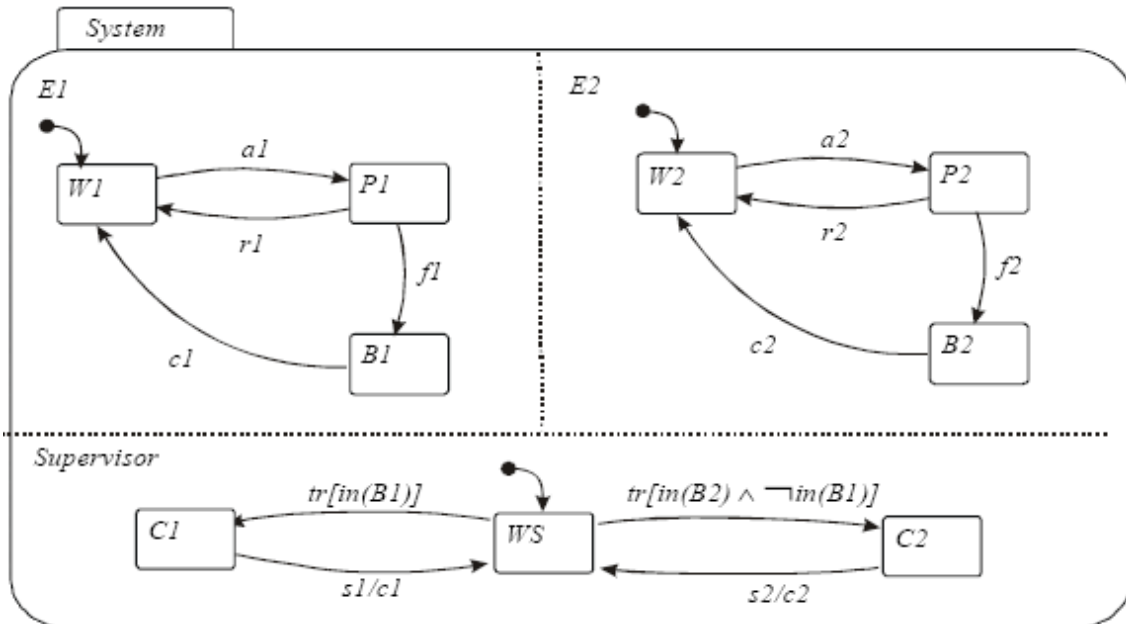


Figure 2.1 - Statecharts representation of equipment with a repairer  
Source : SANTIAGO et al. (2006)

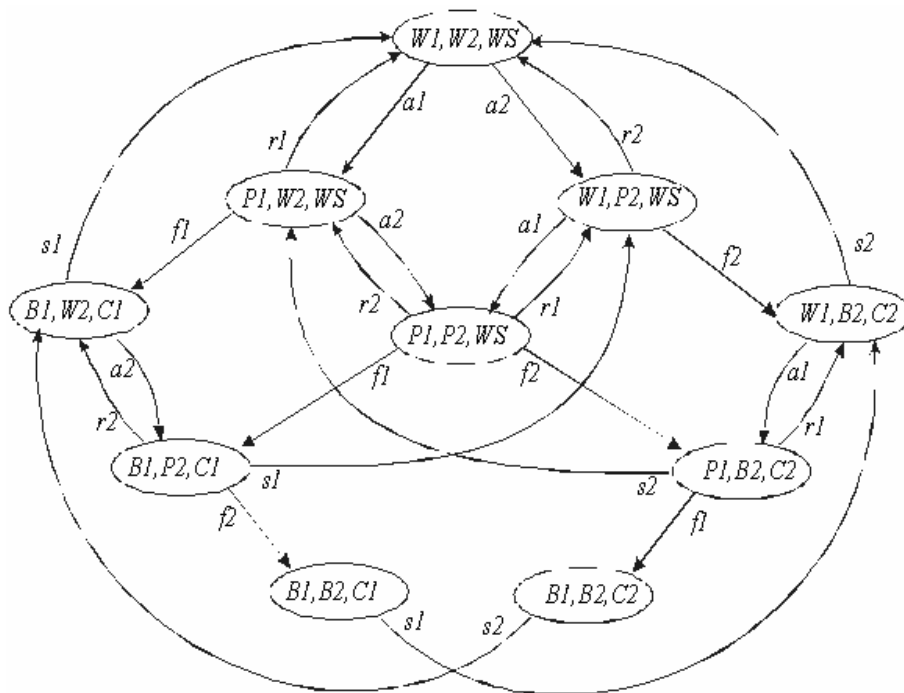


Figure 2.2 - FSM of the example in Figure 2.1

Source : SANTIAGO et al. (2006)

## 2.5 Testing Critical Systems within a Collaborative Scenario

As described before, several systems controlled by software are classified as critical since a failure may cause serious consequences. As examples of critical systems one can mention space applications, navigation systems, banking systems or nuclear plant monitoring. The focus of this work is in space applications that deal with the execution of complex tasks using high cost technologies; consequently, such software demands high quality and testing to guarantee their reliability.

Tests can be applied in different phases of a system development process; even in modeling phases before implementation, it is already possible to fix errors testing a formal specification. These tests based on the software's specification without any knowledge on its internal structure are Functional Tests (or "black box tests").

Nowadays, web offers resources for transmission of data at high speeds in which geographical distance is no longer a critical factor. Thus, the cooperative work among teams located in different places, geographically distant, has become a common trend and even necessary both in business and in academia (TIAN and TAYLOR, 2001). This trend is further enhanced with the concept of globalization. The objective of collaborative systems is helping people involved in a common task supporting communication, coordination and cooperation. The use of such applications means accessibility for any internet user, allows cost saving, time saving, and increases teamwork and efficiency since all manipulated data by one user can be immediately perceived by all other users at remote locations (TIAN and TAYLOR, 2001).

Web-based applications have advantages by offering a low cost solution, since in this architecture, the client can use any operating system and it requires no other proprietary software. Also, nowadays, many people have easy internet access and whenever updates are necessary, this is conducted only in the server where the applications are hosted without any necessity for the users to reinstall any kind of software. So, collaborative web-based systems (also known as E-collaboration) is a common practice adopted for many companies to develop their applications; and in this work a collaborative system was developed implementing classes from an already implemented tool PerformCharts in order to generate “black box tests” for software specifications transmitted by internet.

Collaborative tools can fall into the following categories: Group Document Handling, Real-time conferencing, Non real-time conferencing, Electronic Meeting Systems (EMS) and Electronic Workspace. In case of the tool discussed in this dissertation (WEB-PerformCharts), it belongs to the category of Electronic Workspace due to its main idea in offering teams a common environment for coordination and organization of their work centralizing files and documents in an on-line server (BAFOUTSOU and MENTZAS, 2001). Many



features are commonly found in web-based applications, and those that are relevant for collaborative systems are:

- I. E-mail notifications: to communicate tasks, changes or new activities;
- II. Project management: to control the access level of users and assign tasks to members of a group;
- III. File and document sharing: availability of documents. Particularly in this work, software requirements specifications, software design documents and documentation related to the test process must be available to a group of people involved.

The most common feature in such tools, and at the same time most needed collaboration service, is file and document sharing (BAFOUTSOU and MENTZAS, 2001).

It is usual in space research organizations as INPE and CTA, situations where teams involved in satellite missions are not exactly in one place due to joint collaborations among space agencies to develop space applications. The use of an on-line collaborative tool would definitely aid the software testing activities in this scenario such as the one shown in Figure 2.3.

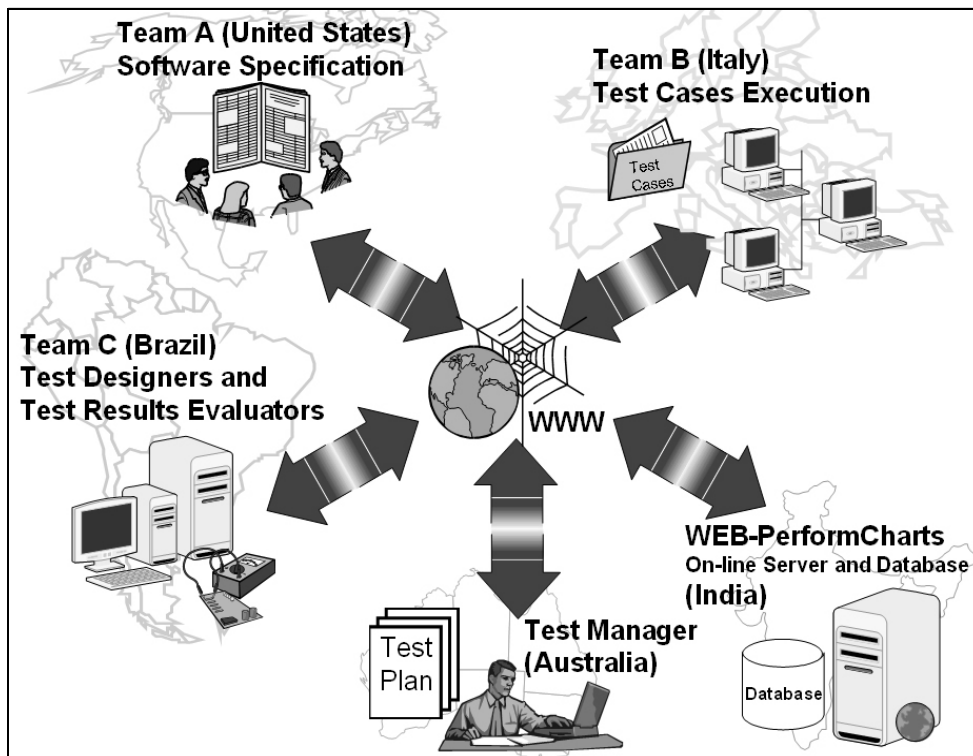


Figure 2.3 - Example of cooperative work

### **3 WEB-PERFORMCHARTS**

#### **3.1 Architecture**

With the objective of enabling different teams, distributed geographically in different locations, working in software testing sharing projects through Internet access, PerformCharts was modified to become WEB-PerformCharts. It is a web-based tool to help software testers working in different places for cooperating in common projects, and using their expertise and know-how in order to benefit software's quality.

PerformCharts tool has been modified to run remotely through a web-based interface and to be hosted in a web server using database access. This on-line database has been implemented in order to promote testers to load and save projects from anywhere to the server, instead of manipulating local files spread over several computers.

Internet development technologies were required for implementation besides the traditional HTML, and the preference was for cost-free technologies such as PHP, MySQL for SQL standard databases and Apache web server software. At the moment, WEB-PerformCharts uses Windows based platform servers; a Linux version is under development.

Once logged in the system, testers are able to create, edit or delete projects and their associated PcML specifications. Each user can manipulate just one project at a time, and when a project is selected (from a list of available projects) it can be modified and the tester can run the test case generation method as many times as required. This is an important feature especially when a software is incorrectly modeled or has to undergo changes in its specification. These changes can be perceived by anyone who can access the same project.

PcML specifications are distributed in projects, that can be created by any user and can be shared among users. The implementation of workflow routines is under study and the communication between them can be integrated with

tester's e-mail addresses. The idea to group users into workgroups seems very useful and will also be studied for implementation.

The number of users who can access WEB-PerformCharts is not limited in theory. It depends directly on the server capacity to support on-line workload as well as on the storage memory. In case of a huge number of users accessing the same server, they could be organized hierarchically according to their functions (e.g. Administrator, User, Guest, Project Manager, General Manager, etc.) providing an easier management. In fact, in this preliminary version, WEB-PerformCharts has two access levels for users: Administrator: full access for any project, and can create another user accounts; User: access just for projects created by her or him. A version control to concurrent access has not yet been developed, but it is expected to be implemented in the near future.

The web-based interface provides the user features to manage her or his projects creating a new one, deleting or modifying an existing project in order to obtain new test cases by running the test case generator method as many times as required. These test cases are stored in an on-line database in the server, and can be accessed anytime by those who have the proper authorization. WEB-PerformCharts uploads a file with PcML specification to web server when user selects it using the provided interface, which is implemented in HTML and PHP. When uploaded, the PcML contents are automatically parsed by a PHP script which extracts any specification data and stores them into a MySQL database. Data inserted in this database is read and used to invoke proper data structures holding the encapsulation, states, events, conditions, parallel components and transitions. It calls appropriate methods from PerformCharts and generates the FSM from its Statecharts specification. If performance evaluation is required, a Markov chain is the result instead of FSM; but in either case (Markov chain or FSM), it is stored in the database and can be extracted in XML format for any other future use.

However, once FSM is available, methods can be applied in order to generate test sequences. WEB-PerformCharts is not planned to be limited to a single

method since the idea is to make these methods as independent cartridges within the system.

### **3.2 Methods for Test Case Generation**

Once the Statecharts representation is converted into a FSM, a test sequence generation method can then be applied on the FSM. Some examples of methods are T-Method, Switch Cover, UIO-Method (Unique Input/Output Sequences) (DERDERIAN et al., 2006) and D-Method (Distinguish Sequences) (SIDHU and LEUNG, 1989). Before developing WEB-PerformCharts, just the Switch Cover method was being used through an integration between PerformCharts and yet another tool Condata (MARTINS et al., 2000) (SANTIAGO et al., 2006) that was developed at UNICAMP. However, although this solution works well for test case generation, is necessary to combine the PerformCharts and Condata tools. In particular Condata only takes the FSM as a base of facts, which is nothing more than a file with FSM described in Prolog source code. This requires another process in converting the output of PerformCharts into the input to Condata becoming a time consuming process, and this is one of the reasons to aggregate a test sequence generator within WEB-PerformCharts. Two such methods were chosen to be incorporated within WEB-PerformCharts: Transition Tour and Switch Cover. Also, WEB-PerformCharts is opened for implementing any other method as long as the method can be applied on an FSM representation. However, it is important to clarify that WEB-PerformCharts still maintains the old approach (in using Condata) through Switch Cover. Therefore, in case a test manager prefers Condata tool, WEB-PerformCharts may still be used. In this case WEB-PerformCharts converts the Statecharts representation into a FSM. Then, the option, in WEB-PerformCharts, of generating the Base of Facts implemented will deliver the necessary Base of Facts required by Condata.

Recollecting, in test sequence generation, users have two alternatives within the WEB-PerformCharts tool. They can use the cartridges from WEB-PerformCharts, or they can export a base of facts which are input to Condata

tool. This conversion is automatically achieved by using a parser written in XSLT. Figure 3.1 describes all basic steps to generate test sequences using WEB-PerformCharts. The generation using one of the methods implemented in WEB-PerformCharts is indicated as “Path A”, and the integration with Switch Cover method from Condata tool as “Path B”.

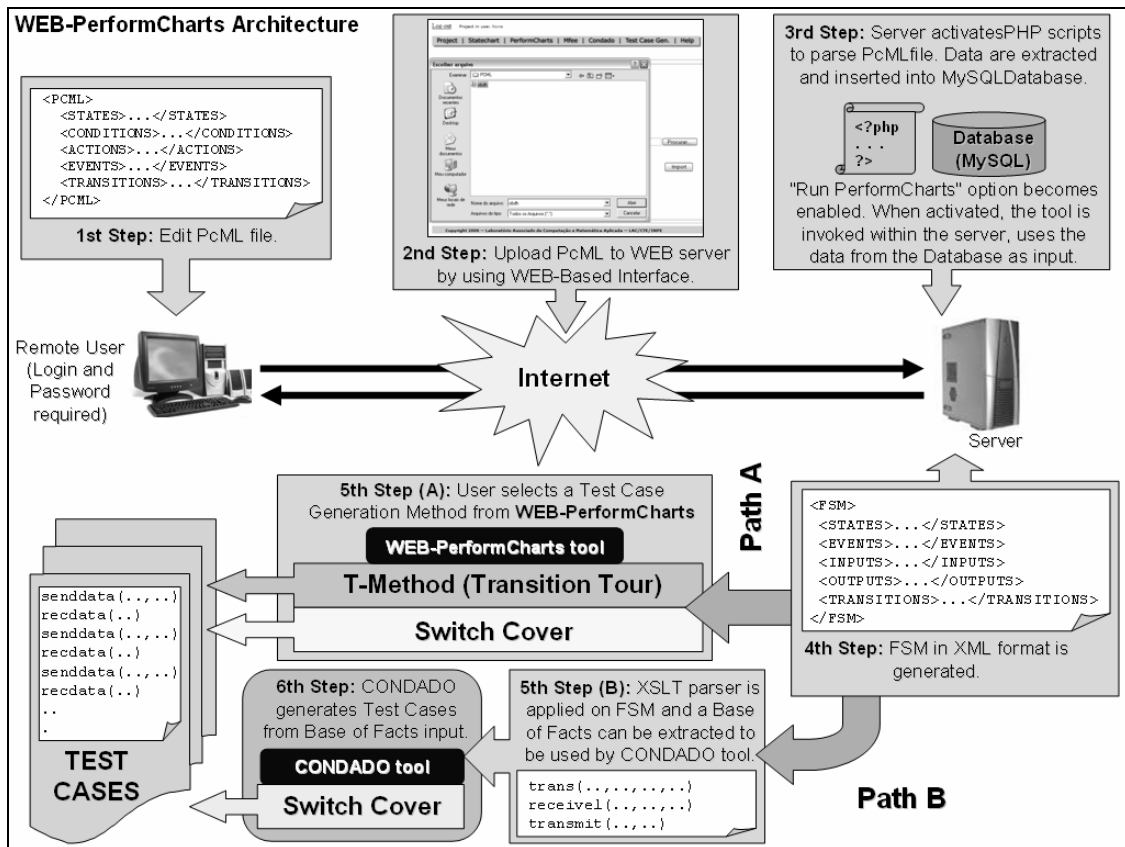


Figure 3.1 - WEB-PerformCharts architecture

### 3.2.1 Transition Tour (T-Method)

Transition Tour is a graph depth search method whose objective is, from an initial state, to traverse all arcs from a graph at least once and return to initial state. This problem is also known as Chinese Postman Problem (DAHURA et al., 1990) whose task is to find a minimized perfect transition tour based on the number of arcs in tour, or on a sum of values related to its respective arcs. The algorithm used in WEB-PerformCharts is shown in Figure 3.2.

```

initial_state = initial state in the graph
current_state = empty
path = empty
number_of_arcs = total number of arcs in the entire graph
arcs_not_traversed = number_of_arcs

WHILE arcs_not_traversed > 0
AND current_state • initial_state DO

    IF arcs_not_traversed = number_of_arcs THEN
        current_state = initial_state
    END IF

    list_of_events = list of events that can be stimulated from
                    current_state

    current_state = new state reached by stimulation of event from
                    list_of_events

    arcs_not_traversed = total number of arcs not traversed in
                        the entire graph

    path = path + event
    DELETE list_of_events

END WHILE

INSERT path into DATABASE

```

Figure 3.2 - Transition Tour algorithm

As can be seen in Figure 3.2, the T-method is relatively simple. In order to traverse a graph, this method requires at least a minimal, strongly connected, and completely specified state machine. However, the sequence may contain some redundant inputs generating loops in the transition tour. In the algorithm implemented in WEB-PerformCharts these redundant inputs were relatively minimized using programming tips that avoid repeated loops, but it does not guarantee an optimal result.

In order to illustrate a transition tour sequence, consider the FSM in Figure 3.2 in which “W” is the initial state. By applying the method, the following sequence is obtained: cfabrbf.

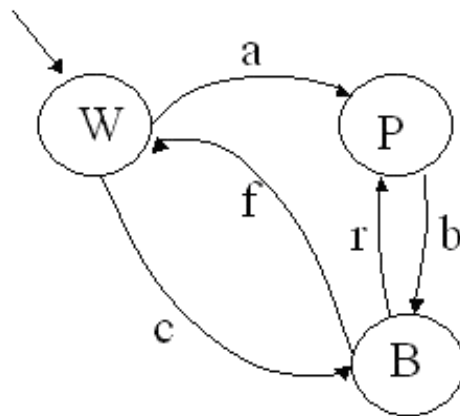


Figure 3.3 - Example of a simple FSM

### 3.2.2 Switch Cover

State transition diagrams are directed graphs. Therefore, concepts and algorithms of graph theory can be used in order to traverse them automatically. The graph theory algorithm implemented in WEB-PerformCharts and Condata is known as sequence of “de Bruijn” (ROBINSON, 1999). Switch Cover method is a more stringent test coverage where a switch is a branch-to-branch pair, and test sequences consist of every branch-to-branch pair from traversed graph. Then, every pair of combinations is executed. Consider the FSM of Figure 3.4 (a) in order to illustrate this algorithm.

1st Step – A dual graph is created from the original one, by converting arcs of the original graph into nodes as shown in Figure 3.4 (b).

2nd Step – For all nodes in the original graph where there is an arc *i* arriving and an arc *j* leaving, an arc is created from *i* to *j* in the dual graph (Figure 3.4 (c)):

3rd Step – The dual graph is transformed into an “Eulerized” graph by balancing the polarity of the nodes. This balance is obtained by duplicating the arcs in such a way that the number of arcs arriving becomes equal to the number of arcs leaving the node. The “Eulerized” graph generated by the tool that corresponds to Figure 3.4 (a) is shown in Figure 3.4 (d).



4th Step – Finally, the nodes are traversed registering those that are visited, generating the following test sequences in WEB-PerformCharts: abrpf, crbf, cf.

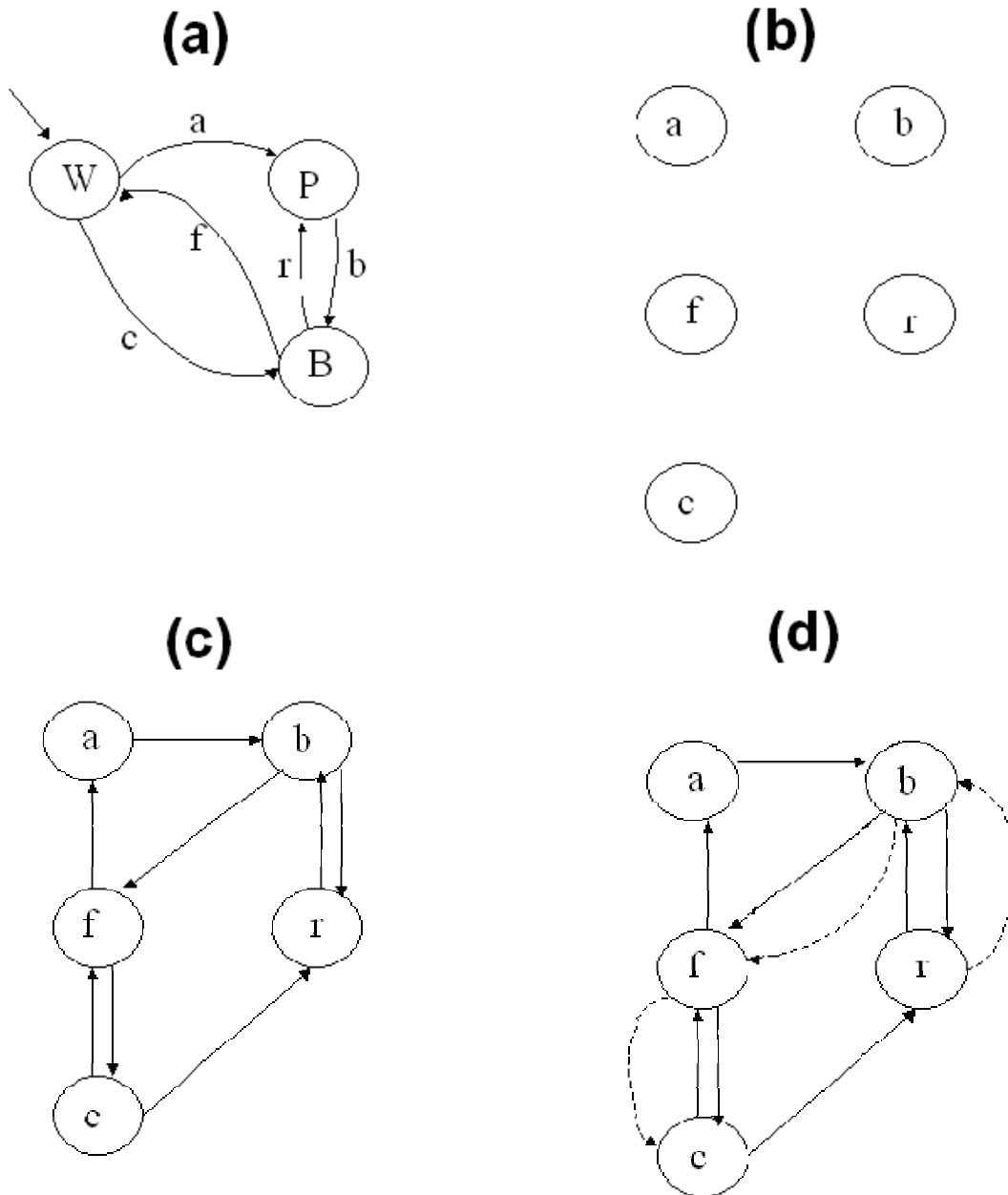


Figure 3.4 - Demonstration of Switch Cover method

Source : Adapted from SANTIAGO et al. (2006)

Condata generated the following sequences: abf, abrpf, cf, crbf. This difference between sequences probably occurs because, in this example, Condata

algorithm uses three more arcs than WEB-PerformCharts during eulerization and consequently generates some more redundant combinations, as it can be seen in Figure 3.5.

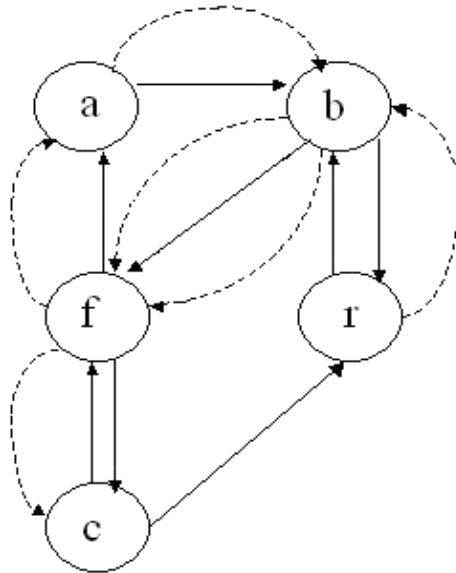


Figure 3.5 - Graph eulerized by Condata tool

Source : Adapted from SANTIAGO et al. (2006)

The Switch Cover algorithm implemented in WEB-PerformCharts, which can be seen in Figure 3.6, uses matching technique to perform graph eulerization and 2-switch set cover (CHOW, 1978) method in order to generate all pairs of combinations between connected transitions. Some programming tips were used in order to avoid loops in sequences and repeated sequences since, depending of the graph complexity, Switch Cover can generate several sequences. One of this tips, for example, is do add a penalty in a sequence when it is repeated, then its execution is cancelled and in a second search the algorithm will avoid this same sequence. However, complex specifications require much extra computational effort and this implementation still does not guarantee an optimal situation, where all branch-to-branch pair of combinations is executed.

```

arcA, arcB = empty
stateA, stateB = empty
// Setting states for dual graph
FOR each arc in graph DO
    SET a new state in dual graph
END FOR
// Tracing arcs between states in dual graph
FOR each pair of arcs in graph DO
    arcA = GET an arc from graph
    arcB = GET another arc from graph
    IF destination of arcA AND source of arcB is in same state THEN
        stateA = GET respective state of arcA
        stateB = GET respective state of arcB
        SET a new arc in dual graph from stateA to stateB
    END IF
END FOR
// Eulerizing graph
signal = empty
list_of_unbalanced = GET all unbalanced states from dual graph
WHILE list_of_unbalanced > 0 DO
    FOR each state in list_of_unbalanced DO
        stateA = GET state from list_of_unbalanced
        signal = GET direction of arc required (incoming or outgoing)
        stateB = GET a state connected with stateA with opposite signal
        IF signal = incoming THEN
            SET new arc from stateB to stateA
        ELSE IF signal = outgoing THEN
            SET new arc from stateA to stateB
        ELSE IF
        END FOR
    list_of_unbalanced = GET all unbalanced states from dual graph
END WHILE
// Generating test sequences
initial_state, current_state = empty
path = empty
test_cases_repository = empty
list_of_events = empty
list_of_initial_states = GET a list of initial states
WHILE there are arcs from each initial state in list_of_initial_states DO
    initial_state = GET state from list_of_initial_states
    WHILE current_state ≠ initial_state DO
        IF path = empty THEN
            current_state = initial_state
        END IF
        list_of_events = events that can be stimulated from current_state
        current_state = new state reached by stimulation of event
        path = path + event
        DELETE list_of_events
        IF current_state = empty THEN
            current_state = initial_state
            DELETE path
            ADD penalty to this path
            DELETE last arc used by path in dual graph
        END IF
    END WHILE
    IF path already exists in test_cases_repository THEN
        DELETE path
        ADD penalty to this path
    ELSE
        ADD path TO test_cases_repository
        DELETE arcs used by path in dual graph
        DELETE path
    END IF
    current_state = empty
END WHILE
INSERT test_cases_repository into DATABASE

```

Figure 3.6 - Switch Cover algorithm

### 3.3 Methodology used in WEB-PerformCharts

In order to show the methodology for test sequence generation through WEB-PerformCharts, consider the example in Figure 2.1. The methodology to generate test cases follows:

1st Step – The system specification in Figure 2.1 is written in PcML. A part of such specification is shown in Figure 3.7.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<PcMl Title="Manufacturing System" Date="2005-08-12">
  <Info>
    <Author>
      <Name>Ana Silvia Martins Serra do Amaral</Name>
      <Email>anasil@lac.inpe.br</Email>
    </Author>
    <Description>
      Project Cote de Resyste - Study Case
      Protocol Conference with variables
    </Description>
  </Info>
  <States>
    <Root Name="System" Type="AND">
      <State Name="E1" Type="XOR" Default="W1">
        .
        .
        .
      </State>
    </Root>
  </States>
  <Conditions>
    <InState Name="Cond1" State="B1"/>
    .
    .
    .
    <NotCondition Name="Cond3" Condition="Cond1"/>
    <ComposedCondition Name="Cond4">
      <ANDCOND Cond1="Cond3" Cond2="Cond2"/>
    </ComposedCondition>
  </Conditions>
  <Actions>
    <EventTriggerAction Name="etaE1" Event="c1"/>
    .
    .
    .
  </Actions>
  <Events>
    <TrueCondition Name="CC1" Condition="Cond1"/>
    .
    .
    .
    <Stochastic Name="a1" Value="5.0"/>
  </Events>
  <Transitions>
    <Transition Source="W1" Event="a1" Destination="P1"/>
    .
    .
    .
  </Transitions>
</PcMl>
```

Figure 3.7 - PcML specification of modeling in Figure 2.1

2nd Step – WEB-PerformCharts is accessed and PcML file is uploaded to Web server through the user interface shown in Figure 3.8.

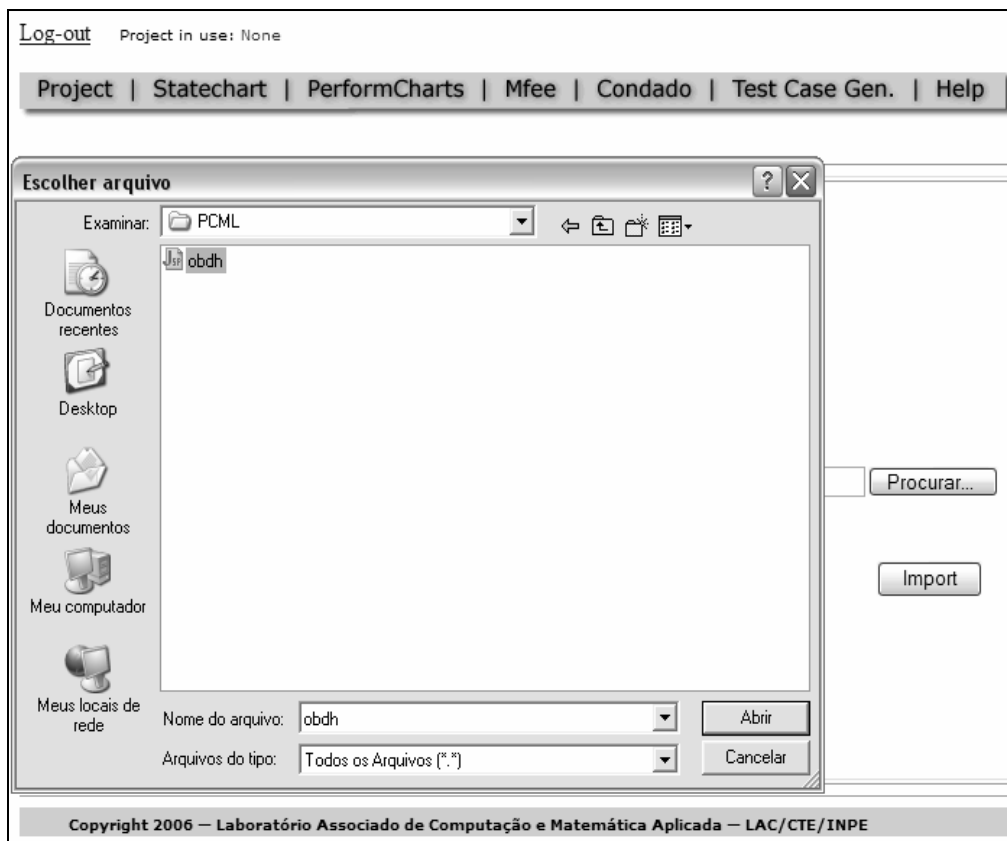


Figure 3.8 - Upload interface in web server

3rd Step – PcML file is automatically parsed by PHP and data are inserted into a MySQL database. “Run PerformCharts” option is enabled and generates a FSM from Statecharts specification. The FSM of the example shown in Figure 2.1 is illustrated in Figure 2.2.

4th Step – FSM data is included into database and can be extracted as an XML file. Part of this file can be seen in Figure 3.9. Once FSM is obtained, tester can generate test sequences using Transition Tour or Switch Cover methods available within WEB-PerformCharts (Path A), or export a file with a suitable input to Condata tool to run independently (Path B) from the WEB-PerformCharts tool. Both paths have been tested.

```

<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="mfeeX.xsl"?>
<MFEE>
  <STATES>
    <STATE NAME="W1W2WS" TYPE="inicial"/>
    <STATE NAME="P1W2WS" TYPE="normal"/>
    . . .
    <STATE NAME="W1W2WS" TYPE="final"/>
  </STATES>
  <EVENTS>
    <EVENT NAME="a1" VALUE="1"/>
    . . .
  </EVENTS>
  <INPUTS>
    <INPUT EVENT="a1"/>
    . . .
  </INPUTS>
  <OUTPUTS>
    <OUTPUT EVENT="c1"/>
    . . .
  </OUTPUTS>
  <TRANSITIONS>
    <TRANSITION SOURCE="W1W2WS" DESTINATION="P1W2WS">
      <INPUT INTERFACE="L">a1</INPUT>
      <OUTPUT></OUTPUT>
    </TRANSITION>
    . . .
  </TRANSITIONS>
</MFEE>

```

Figure 3.9 - FSM specified in XML

5th Step (A) – Transition Tour method was applied to the generated FSM and, this graph consisting of 10 states and 24 transition arcs was entirely covered using 49 steps. When a test sequence is generated, it is inserted into database. The full test sequence is shown in Table 3.1.

In other words, Table 3.1 shows that, the sequence of events run by Transition Tour method in WEB-PerformCharts was: a2, f2, s2, a1, a2, f2, etc.

Table 3.1 - Test Cases generated by Transition Tour method

Step	Event	State
1	A2	W1P2WS
2	F2	W1B2C2
3	S2	W1W2WS
4	A1	P1W2WS
5	A2	P1P2WS
6	F2	P1B2C2
7	S2	P1W2WS
8	F1	B1W2C1
9	S1	W1W2WS
10	A2	W1P2WS
11	r2	W1W2WS
12	a1	P1W2WS
13	r1	W1W2WS
14	a2	W1P2WS
15	a1	P1P2WS
16	r2	P1W2WS
17	a2	P1P2WS
18	f1	B1P2C1
19	s1	W1P2WS
20	f2	W1B2C2

Step	Event	State
21	a1	P1B2C2
22	f1	B1B2C2
23	s2	B1W2C1
24	a2	B1P2C1
25	f2	B1B2C1
26	s1	W1B2C2
27	s2	W1W2WS
28	a1	P1W2WS
29	f1	B1W2C1
30	s1	W1W2WS
31	a2	W1P2WS
32	r2	W1W2WS
33	a1	P1W2WS
34	r1	W1W2WS
35	a2	W1P2WS
36	a1	P1P2WS
37	r1	W1P2WS
38	f2	W1B2C2
39	a1	P1B2C2
40	r1	W1B2C2

Step	Event	State
41	s2	W1W2WS
42	a1	P1W2WS
43	a2	P1P2WS
44	f2	P1B2C2
45	s2	P1W2WS
46	f1	B1W2C1
47	a2	B1P2C1
48	r2	B1W2C1
49	s1	W1W2WS

Switch Cover method also was applied to the generated FSM and, this graph was entirely covered using 73 steps. The moment sequences are generated, they are inserted into database. The set of 12 test sequences is shown in Table 3.2.

In other words, Table 3.2 shows that, the sequence of events run by Switch Cover method in WEB-PerformCharts was: a1, r1; a1, f1, a2, etc. Table lines filled with hyphen means end of a sequence and beginning of another.

All information presented (PcML specifications, FSM, and test cases) are shared by any logged user in WEB-PerformCharts since they are totally stored into an on-line database and can be accessed in real-time conditions.

Table 3.2 - Test Cases generated by Switch Cover method

Step	Event	State
1	a1	P1W2WS
2	r1	W1W2WS
-	-	-
3	a1	P1W2WS
4	f1	B1W2C1
5	a2	B1P2C1
6	r2	B1W2C1
7	a2	B1P2C1
8	f2	B1B2C1
9	s1	W1B2C2
10	a1	P1B2C2
11	r1	W1B2C2
12	a1	P1B2C2
13	f1	B1B2C2
14	s2	B1W2C1
15	a2	B1P2C1
16	s1	W1P2WS
17	a1	P1P2WS
18	r1	W1P2WS
19	a1	P1P2WS
20	f1	B1P2C1
21	r2	B1W2C1
22	s1	W1W2WS
-	-	-
23	a1	P1W2WS
24	a2	P1P2WS
25	r1	W1P2WS
26	R2	W1W2WS
-	-	-

Step	Event	State
27	a1	P1W2WS
28	a2	P1P2WS
29	f1	B1P2C1
30	f2	B1B2C1
31	s1	W1B2C2
32	s2	W1W2WS
-	-	-
33	a1	P1W2WS
34	a2	P1P2WS
35	r2	P1W2WS
36	r1	W1W2WS
-	-	-
37	a1	P1W2WS
38	a2	P1P2WS
39	f2	P1B2C2
40	r1	W1B2C2
41	s2	W1W2WS
-	-	-
42	a2	W1P2WS
43	a1	P1P2WS
44	r2	P1W2WS
45	f1	B1W2C1
46	s1	W1W2WS
-	-	-
47	a2	W1P2WS
48	r2	W1W2WS
-	-	-
49	a2	W1P2WS
50	f2	W1B2C2

Step	Event	State
51	a1	P1B2C2
52	s2	P1W2WS
53	r1	W1W2WS
-	-	-
54	a2	W1P2WS
55	a1	P1P2WS
56	f2	P1B2C2
57	f1	B1B2C2
58	s2	B1W2C1
59	s1	W1W2WS
-	-	-
60	a2	W1P2WS
61	a1	P1P2WS
62	f1	B1P2C1
63	s1	W1P2WS
64	r2	W1W2WS
-	-	-
65	a2	W1P2WS
66	a1	P1P2WS
67	f2	P1B2C2
68	s2	P1W2WS
69	f1	B1W2C1
70	a2	B1P2C1
71	s1	W1P2WS
72	f2	W1B2C2
73	s2	W1W2WS
-	-	-



5th Step (B) – WEB-PerformCharts has an option “Get base of facts” that must be accessed in order to call an integrated XSLT parser. This parser is responsible in converting the XML data of FSM into the required input for Condata tool to generate test sequences. A part of this input is in Figure 3.10. Condata tool is implemented in Prolog and hence it requires the input as a base of facts.

```
inicial( estado0 ).

trans( estado0, transicao0, estado2, L0, Ln ) :-
    receivel( 'a2', L0, L1 ),
    transmit( L1, Ln ).

trans( estado2, transicao1, fim, L0, Ln ) :-
    receivel( 'r2', L0, L1 ),
    transmit( L1, Ln ).

trans( estado0, transicao2, estado1, L0, Ln ) :-
    receivel( 'a1', L0, L1 ),
    transmit( L1, Ln ).

trans( estado1, transicao3, fim, L0, Ln ) :-
    receivel( 'r1', L0, L1 ),
    transmit( L1, Ln ).

trans( estado2, transicao4, estado3, L0, Ln ) :-
    receivel( 'a1', L0, L1 ),
    transmit( L1, Ln ).

trans( estado3, transicao5, estado2, L0, Ln ) :-
    receivel( 'r1', L0, L1 ),
    transmit( L1, Ln ).

trans( estado1, transicao6, estado3, L0, Ln ) :-
    receivel( 'a2', L0, L1 ),
    transmit( L1, Ln ).

...
..
.
```

Figure 3.10 - Base of Facts for Condata tool

6th Step – As Condata tool is not directly integrated in WEB-PerformCharts, the base of facts extracted in previous step must be saved as a text file and taken to a computer which Condata tool is installed. When Condata runs with base of facts as input, it applies Switch Cover method to obtain test sequences. Test

sequences generated by Condata tool are obtained locally and, therefore they are not stored in WEB-PerformCharts database.

## 4 RESULTS

In order to demonstrate and compare results obtained by Transition Tour and Switch Cover methods implemented within WEB-PerformCharts, seven case studies are shown along with a set of following attributes:

- I. Time to Generate Test Cases: It is the time (in seconds) spent by the method in order to generate test cases. It is known that many factors can affect time spent to generate test cases such as workload of the computer at the same moment while method is processing; therefore, all case studies were executed three times and the best mark was taken;
- II. Number of Test Cases: Number of test cases, or test sequences, generated by each method. Transition Tour always generates just one test sequence; however Switch Cover can generate one or many of them with varied number of steps;
- III. Size of Test Cases: It is the number of events needed to be applied in the graph in order to generate one or several test sequences. In case of Switch Cover method, the smallest and bigger ones are listed;
- IV. Size of FSM Examined: Describes number of nodes and arcs in the FSM examined for method. It is an interesting information since Switch Cover method converts initial FSM into a dual graph making, in most of the cases, a much bigger FSM. The number in parentheses represents the number of graph arcs after eulerization;
- V. Total Number of Events: The total number of events needed to perform all test cases generated.

### 4.1 First Case Study

A simple example that was already presented in Figure 3.3 is explored. This example consists of a basic FSM composed of 3 states and 5 transitions. Some results obtained by WEB-PerformCharts running both the implemented methods

can be seen in Table 4.1. Since this is already a FSM, no conversion from Statecharts was required.

Table 4.1 - Results obtained in first case study

Method / Results	Transition Tour	Switch Cover
<b>Time to Generate Test Cases</b>	1s	9s
<b>Number of Test Cases</b>	1 case	3 cases
<b>Size of Test Cases</b>	7 events	Between 2 and 6 events
<b>Size of FSM Examined</b>	3 nodes, 5 arcs	5 nodes, 8 (11) arcs
<b>Total of Events</b>	7 events	11 events

Test cases obtained by Transition Tour and Switch Cover methods are in Sections 3.2.1 and 3.2.2 respectively.

## 4.2 Second Case Study

This example is a simulation of a manufacturing system with a repairer. Its detailed explanation is in Section 2.4, and its Statecharts representation is showed in Figure 2.1. It is composed of 9 states and 12 transitions and can be considered as a medium complex case study.

After the conversion from Statecharts (Figure 2.1) to FSM (Figure 2.2), Transition Tour and Switch Cover methods were applied generating results showed in Table 4.2.

Table 4.2 - Results obtained in second case study

Method / Results	Transition Tour	Switch Cover
<b>Time to Generate Test Cases</b>	1s	13s
<b>Number of Test Cases</b>	1 case	12 cases
<b>Size of Test Cases</b>	49 events	Between 2 and 20 events
<b>Size of FSM Examined</b>	10 nodes, 24 arcs	24 nodes, 60 (80) arcs
<b>Total of Events</b>	49 events	71 events

In case of Switch Cover method, a 24 arc FSM was converted into a dual graph with 60 arcs. Another 20 arcs were added to the graph in the eulerization

process resulting in 80 arcs examined. Table 3.1 and 3.2 shows, respectively, all test cases obtained by Transition tour and Switch Cover methods.

### 4.3 Third Case Study

It is a behavioral representation of TCP protocol (AMARAL, 2005) which is largely implemented in many internet applications as Telnet, FTP and SMTP. TCP is responsible to establish a connection between computers delivering and ensuring integrity of data packages. The connection establishment process is named as three-way handshake, and the abstraction level of this model is high since it does not represent data treatment processes. Protocol may pack and send user data; at the same moment data is sent, it may keep a timer waiting response from an *ACK*; receiver computer may rearrange data, check duplicity and calculate checksum. These details related to data treatment are encapsulated within *Established\_CL* and *Established\_SV* states. Figure 4.1 shows Statecharts specification of this case study, while Figure 4.2 is its FSM generated and results overview are in Table 4.3.

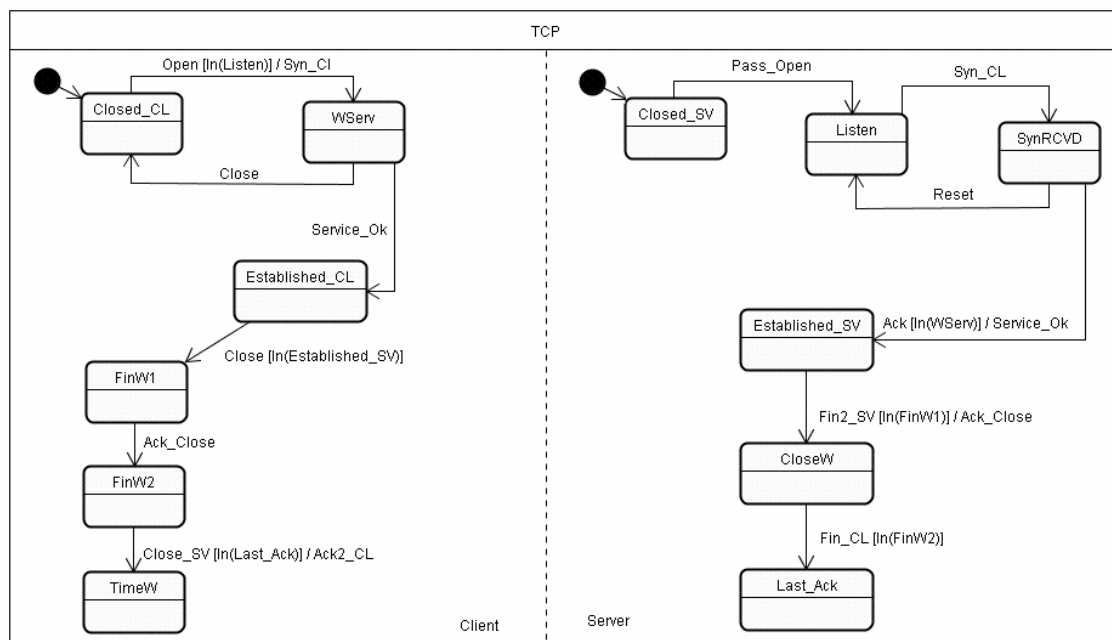


Figure 4.1 - Statecharts representation of TCP protocol behavior

Source : AMARAL (2005)

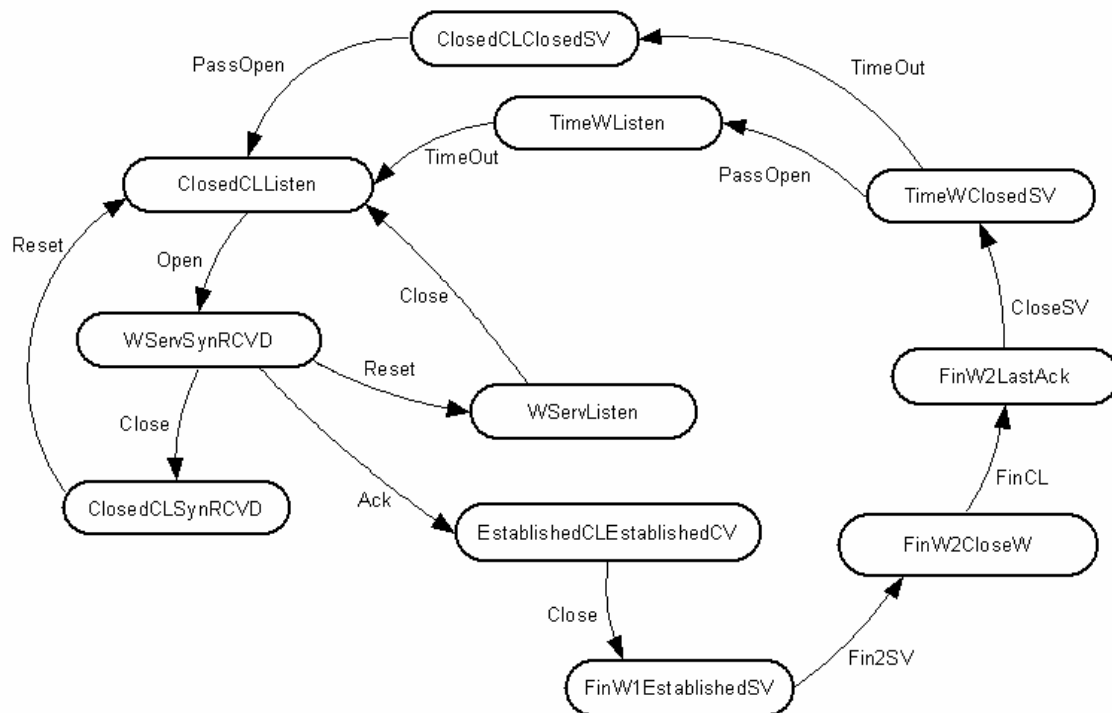


Figure 4.2 - FSM generated from Figure 4.1

Table 4.3 - Results obtained in third case study

Method / Results	Transition Tour	Switch Cover
<b>Time to Generate Test Cases</b>	2s	12s
<b>Number of Test Cases</b>	1 case	1 case
<b>Size of Test Cases</b>	22 events	22 events
<b>Size of FSM Examined</b>	11 nodes, 14 arcs	14 nodes, 17 (22) arcs
<b>Total of Events</b>	22 events	22 events

It is interesting to observe that, in this case, FSM did not require much more arcs to be eulerized (just 5) and, moreover as this graph was arranged in such a way, the algorithm found a complete path easily. Therefore, the number of arcs included by eulerization was not enough to generate more than one complete test case (which must have an initial state as source and destination), and besides, both sequences has the same number of events (22), but followed different paths as can be seen in Table 4.4 and Table 4.5.

Table 4.4 - Test cases from Transition Tour in third case study

Step	Event	State
1	PassOpen	ClosedCLListen
2	Open	WServSynRCVD
3	Ack	EstablishedCLEstablishedSV
4	Close	FinW1EstablishedSV
5	Fin2SV	FinW2CloseW
6	FinCL	FinW2LastAck
7	CloseSV	TimeWClosedSV
8	PassOpen	TimeWListen
9	Timeout	ClosedCLListen
10	Open	WServSynRCVD
11	Reset	WServListen
12	Close	ClosedCLListen
13	Open	WServSynRCVD
14	Close	ClosedCLSynRCVD
15	Reset	ClosedCLListen
16	Open	WServSynRCVD
17	Ack	EstablishedCLEstablishedSV
18	Close	FinW1EstablishedSV
19	Fin2SV	FinW2CloseW
20	FinCL	FinW2LastAck
21	CloseSV	TimeWClosedSV
22	Timeout	ClosedCLClosedSV

Table 4.5 - Test cases from Switch Cover in third case study

Step	Event	State
1	PassOpen	ClosedCLListen
2	Open	WServSynRCVD
3	Close	ClosedCLSynRCVD
4	Reset	ClosedCLListen
5	Open	WServSynRCVD
6	Reset	WServListen
7	Close	ClosedCLListen
8	Open	WServSynRCVD
9	Ack	EstablishedCLEstablishedSV
10	Close	FinW1EstablishedSV
11	Fin2SV	FinW2CloseW
12	FinCL	FinW2LastAck
13	CloseSV	TimeWClosedSV
14	PassOpen	TimeWListen
15	Timeout	ClosedCLListen
16	Open	WServSynRCVD
17	Ack	EstablishedCLEstablishedSV
18	Close	FinW1EstablishedSV
19	Fin2SV	FinW2CloseW
20	FinCL	FinW2LastAck
21	CloseSV	TimeWClosedSV
22	Timeout	ClosedCLClosedSV



#### 4.4 Fourth Case Study

This specification simulates the behavior of a classical Producer-Consumer problem (AMARAL, 2005) largely studied in programming, mainly in the area of operating systems. It consists basically of three parallel components: Producer (Produtor), Consumer (Consumidor), and a buffer (Buffer) that is incremented by Producer and decremented by Consumer. The Statecharts specification for this case is shown in Figure 4.3, while its FSM generated is in Figure 4.4. Table 4.3 shows results overview.

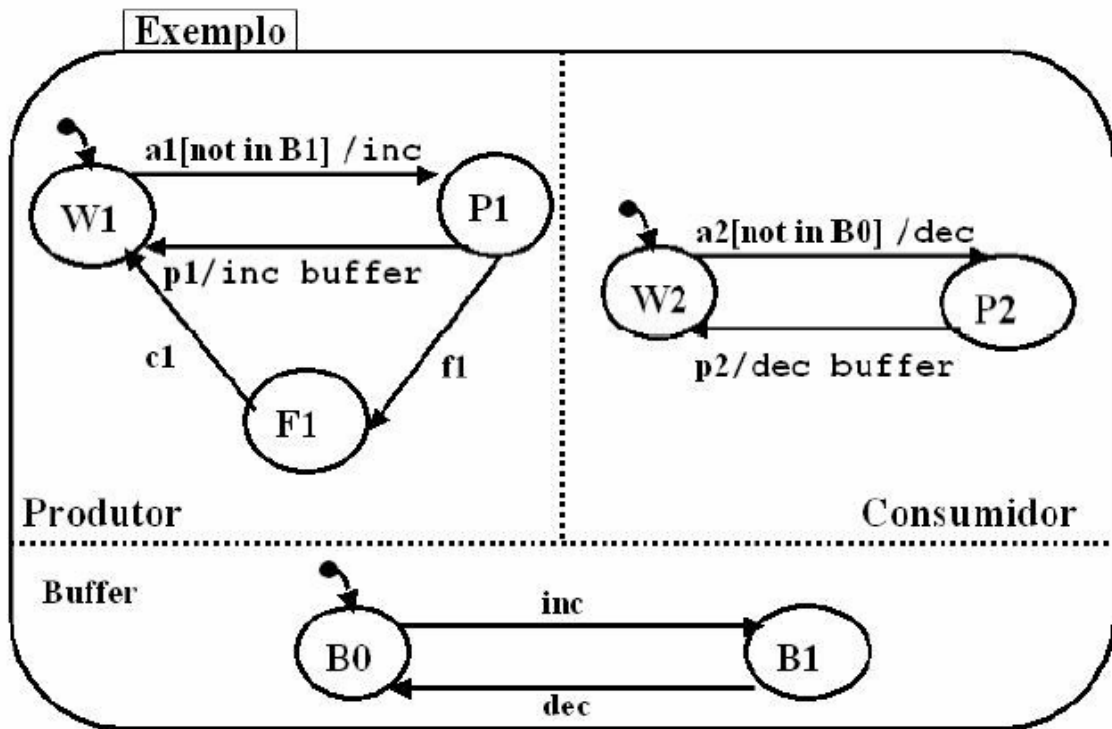


Figure 4.3 - Statecharts representation of Producer-Consumer problem

Source : AMARAL (2005)

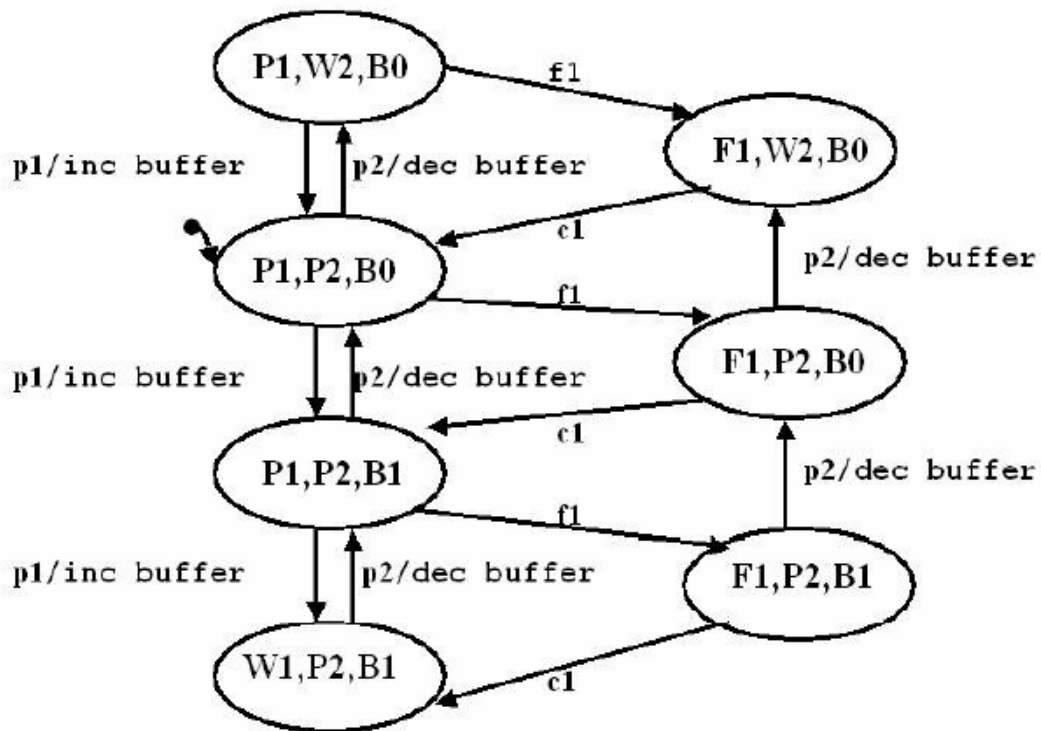


Figure 4.4 - FSM generated from Figure 4.3

Source : AMARAL (2005)

Table 4.6 - Results obtained in fourth case study

Method / Results	Transition Tour	Switch Cover
Time to Generate Test Cases	1s	10s
Number of Test Cases	1 case	8 cases
Size of Test Cases	31 events	Between 2 and 9 events
Size of FSM Examined	7 nodes, 14 arcs	14 nodes, 30 (42) arcs
Total of Events	31 events	35 events

Figures 4.3 and 4.4 show, respectively, that Statecharts specification for this example has 7 states and 8 transitions, and its FSM has 7 nodes and 14 arcs. This case is different from other case studies where there are outputs to be generated while traversing the FSM. Generation of outputs is a resource implemented in WEB-PerformCharts that was brought from PerformCharts, and is used in order to aid tester to verify whether a simulated event works and if it did what are the expected effects, i.e. not only the expected destination state

but also the expected output. Table 4.7 and Table 4.8 show results obtained for this case using Transition Tour and Switch Cover methods respectively. Note that, for this case, a new column named as output is added in order to represent outputs performed by its respected events.

Table 4.7 - Test cases from Transition Tour in fourth case study

Step	Event	State	Output
1	p2	P1W2B0	
2	f1	F1W2B0	
3	c1	P1P2B0	
4	f1	F1P2B0	
5	p2	F1W2B0	O2
6	c1	P1P2B0	
7	p1	P1P2B1	O1
8	p2	P1P2B0	O2
9	p2	P1W2B0	O2
10	p1	P1P2B0	O1
11	f1	F1P2B0	
12	c1	P1P2B1	
13	f1	F1P2B1	
14	p2	F1P2B0	O2
15	p2	F1W2B0	O2
16	c1	P1P2B0	

Step	Event	State	Output
17	p1	P1P2B1	O1
18	p1	W1P2B1	O1
19	p2	P1P2B1	O2
20	p2	P1P2B0	O2
21	p2	P1W2B0	O2
22	f1	F1W2B0	
23	c1	P1P2B0	
24	f1	F1P2B0	
25	c1	P1P2B1	
26	f1	F1P2B1	
27	c1	W1P2B1	
28	p2	P1P2B1	O2
29	p1	W1P2B1	O1
30	p2	P1P2B1	O2
31	p2	P1P2B0	O2

Table 4.8 - Test cases from Switch Cover in fourth case study

Step	Event	State	Output
1	p1	P1P2B1	
2	p1	W1P2B1	O1
3	p2	P1P2B1	O2
4	p1	W1P2B1	O1
5	p2	P1P2B1	O2
6	f1	F1P2B1	
7	c1	W1P2B1	
8	p2	P1P2B1	O2
9	p2	P1P2B0	O2
-	-	-	-
10	p1	P1P2B1	O1
11	f1	F1P2B1	
12	p2	F1P2B0	O2
13	c1	P1P2B1	
14	p1	W1P2B1	O1
15	p2	P1P2B1	O2
16	p2	P1P2B0	O2
-	-	-	-
17	p1	P1P2B1	O1
18	p2	P1P2B0	O2
-	-	-	-

Step	Event	State	Output
19	f1	F1P2B0	
20	c1	P1P2B1	
21	f1	F1P2B1	
22	p2	F1P2B0	O2
23	p2	F1W2B0	O2
24	c1	P1P2B0	
-	-	-	-
25	f1	F1P2B0	
26	p2	F1W2B0	O2
27	c1	P1P2B0	
-	-	-	-
28	f1	F1P2B0	
29	c1	P1P2B1	
30	p2	P1P2B0	O2
-	-	-	-
31	p2	P1W2B0	O2
32	p1	P1P2B0	O1
-	-	-	-
33	p2	P1W2B0	O2
34	f1	F1W2B0	
35	c1	P1P2B0	
-	-	-	-

## 4.5 Fifth Case Study

This is a medium complex case. It is a piece of APEX (SANTIAGO et al., 2008) software which is an astrophysical experiment aboard on a Brazilian scientific satellite; more precisely, this is a command recognition component. Command messages are sent in a format composed of six fields: SYNC (EB9 synchronization value), EID (experiment identification), TYPE (specifies accepted commands), SIZE (amount of bytes in the DATA field), DATA and CKSUM (8-bit checksum). SIZE and DATA fields are optional and depend on the type of command.

The behavior of command recognition component software is shown in Figure 4.5 and it is a low-level modeling since it is possible to see all the specified values of the protocol frame fields.

In Statecharts, the initial configuration is (*Idle, Waiting Sync*) and all fields of the command message are verified by the experiment through on-board software. For instance, assume that a command sent to the experiment with these values was received exactly as it was sent (i.e. no data corruption during the command transmission):  $SYNC = EB9$ ,  $EID = 2$ ,  $TYPE = 03$ ,  $CKS = 80$ .

Occurrence of event *EB9* makes the *B XOR*-state to change its active sub-state from *Waiting Sync* to *Checking Field*. *Waiting Expld* is the initial state of *Checking Field*. Then, the action (internal event) starting timing counting makes the *A XOR*-state change from sub-state *Idle* to sub-state *Counting Time*. After *EB9*, the *eid rc[eid = 2]* event is triggered because  $EID = 2$ . The *B XOR*-state moves from *Waiting Expld* to *Waiting Type*. As  $TYPE = 03$ , the event type *rc [type >=01 and type <= 05]* is triggered changing from *Waiting Type* to *Waiting Checksum*. Finally, as the value of checksum received was correct, the event *cksum rc[cksum OK]* is triggered and the *B XOR*-state switches from *Waiting Checksum* to *Waiting Sync* in order to wait for another command. Also, the action (internal event) *command received* means the message was received and accepted by the experiment software. The *A XOR*-state will change from

*Counting Time to Idle* when this event is fired. With this action, the timing counting is interrupted and the software is enabled to receive another command.

If the software detects errors in any of the fields of the command message, the communication is aborted, the command is discarded and the experiment remains ready to receive a new command.

A timeout mechanism is implemented in both the computers (state *A* for the experiment software). After receiving an *EB9* event, the Implementation Under Test (IUT) must start counting the time. For example, if the time defined for receiving an entire command from the On-Board Data Handler (OBDH) elapses, for this particular protocol specification this period is 500 ms, *waiting time expired [not in (Aborting)]* event is triggered in *A* state and the *B* state will change from *Checking Field* to *Waiting Sync* due to the action (internal event) timeout. This makes the system to return to the initial configuration.

Figure 4.6 shows the FSM generated from this specification in Statecharts, and Table 4.9 shows results obtained by WEB-PerformCharts generating test cases for this application.

Table 4.9 - Results obtained in fifth case study

Method / Results	Transition Tour	Switch Cover
<b>Time to Generate Test Cases</b>	2s	7s
<b>Number of Test Cases</b>	1 case	7 cases
<b>Size of Test Cases</b>	70 events	Between 1 and 6 events
<b>Size of FSM Examined</b>	6 nodes, 12 arcs	12 nodes, 24 (249) arcs
<b>Total of Events</b>	70 events	27 events

This case is an unusual example where the total number of events needed by Switch Cover to traverse FSM (27) is much less than Transition Tour (70), proving that definitely there is no unique technique that is capable in dealing with all possible situations since usually the opposite (Transition Tour fewer than Switch Cover) happens, as can be seen in other case studies.

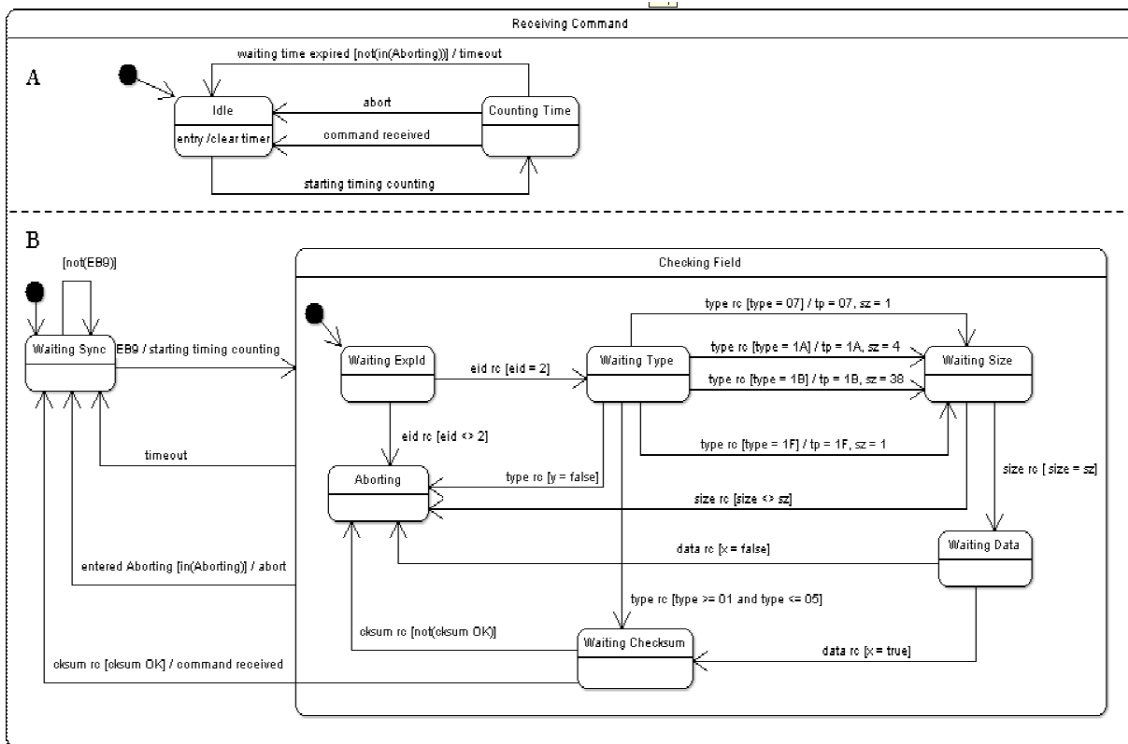


Figure 4.5 - Statecharts representation of APEX system

Source : SANTIAGO et al. (2008)

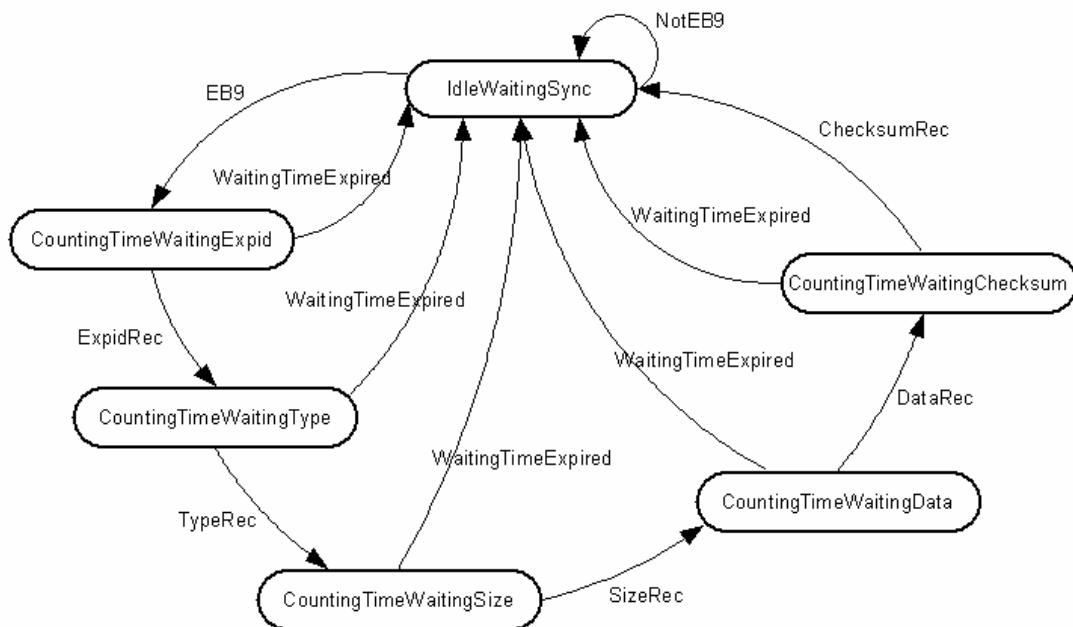


Figure 4.6 - FSM generated from Figure 4.5

## 4.6 Sixth Case Study

This high complex software behavior model was specified in the scope of the *Qualidade do Software Embarcado em Aplicações Espaciais* (QSEE - Quality of Space Application Embedded Software) research project (SANTIAGO et al., 2007). This project is an experience at INPE in outsourcing the development of satellite payload embedded software. The software, SWPDC, is in charge of collecting and formatting data from Event Pre-Processors (EPPs), receiving and executing commands from the OBDH computer, transmitting telemetry data to the OBDH, generating housekeeping information, accomplishing data memory management, implementing fault tolerance mechanisms and supporting loading of new programs on the fly. EPPs are front-end processors in charge of fast data processing of X-ray cameras signals of an astrophysical scientific experiment under development at INPE and the OBDH is the satellite platform computer (SANTIAGO et al., 2007).

A project like QSEE fits very well into a collaborative systems approach. Taking into account only the on-board computers, it is perfectly possible that different organizations might be in charge of distinct computing subsystems development. For instance, one organization may be responsible for developing the OBDH, and its related software, another for the SWPDC computer, and the SWPDC itself, and even another for the EPPs and associated software. A completely distinct organization may be in charge of Verification and Validation of these software in an Independent approach, known as Independent Verification and Validation (IVV) (SANTIAGO et al., 2007). In such scenario, WEB-PerformCharts comes into aid IVV's test designers to generate test cases remotely via web.

Statecharts shown in Figure 4.7 is just a small part of the entire SWPDC modeling. It deals only with some state management of the software. *Managing State* is an *AND* state composed of four *XOR* states, denoted *A*, *B*, *C* and *D*. *A* and *B* are sub-states wondering if EPPs are active and able to send data collected during their operation or if they are inactive. Sub-state *C* models event



report generation to be included in Housekeeping data to be sent to the OBDH. Housekeeping data have status information related to the health not only of SWPDC but also of the hardware of the computing subsystem. Sub-state *D* is related to data acquisition from EPPs by SWPDC computer. EPPs can generate three types of data known as Scientific, Diagnosis and Test data. So, SWPDC shall be able to interact with EPPs in order to request these data. In Figure 4.7, DD stands for Data-Diagnosis type, DT means Data-Test type, HK means Data-Housekeeping type and DM means Data-Dump type. For instance, *prepare\_DT* event instructs SWPDC to acquire Test data from EPPs to be transmitted later to the OBDH.

FSM generated from this specification is huge (40 states and 304 transitions) and therefore it has not been included. However, the results are showed in Table 4.10.

Table 4.10 - Results obtained in sixth case study

Method / Results	Transition Tour	Switch Cover
<b>Time to Generate Test Cases</b>	3s	193s
<b>Number of Test Cases</b>	1 case	193 cases
<b>Size of Test Cases</b>	1046 events	Between 2 and 267 events
<b>Size of FSM Examined</b>	40 nodes, 304 arcs	304 nodes, 2096 (6930) arcs
<b>Total of Events</b>	1046 events	5501 events

In this case, it is more evident that the graph eulerization spends most of the time to generate test cases for Switch Cover method, since its FSM increases three times after 4834 new arcs are added. It resulted into a graph with 6930 arcs (originally 2096 arcs). The same did not occur with Transition Tour that executed in a very less time to generate test cases, as expected.

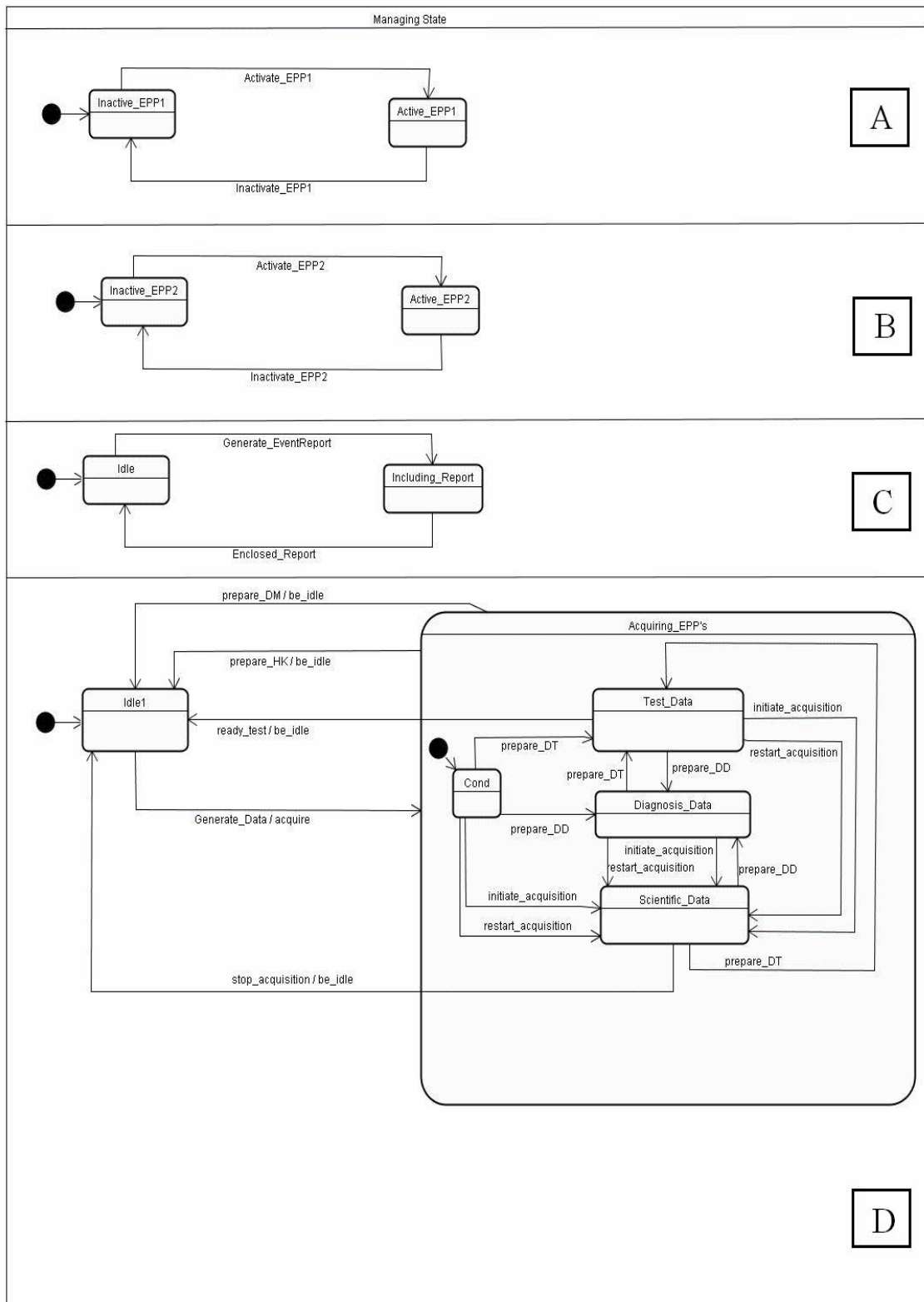


Figure 4.7 - Statecharts representation of OBDH system

Source : SANTIAGO et al. (2007)

#### 4.7 Seventh Case Study

As the most complex example, this specification refers to the same APEX (INPE, 1998) software in the fifth case study. The difference is with respect to the level of details, since this specification models explicitly several values that can be assigned to fields within the Command messages. It results in a specification composed by much more transitions, as can be seen in Figure 4.8.

Table 4.11 shows results obtained by WEB-PerformCharts generating test cases for this application.

Table 4.11 - Results obtained in seventh case study

<b>Method / Results</b>	<b>Transition Tour</b>	<b>Switch Cover</b>
<b>Time to Generate Test Cases</b>	2s	3375s
<b>Number of Test Cases</b>	1 case	166 cases
<b>Size of Test Cases</b>	2275 events	Between 2 and 2811 events
<b>Size of FSM Examined</b>	19 nodes, 69 arcs	69 nodes, 199 (8059) arcs
<b>Total of Events</b>	2275 events	5497 events

This case is an interesting analysis because if just the number of arcs in FSM is compared, this case (69 arcs) seems less complex than previous (304 arcs). However, graph complexity is not related to this number of arcs but in its number of paths. And this must be the reason that, in practice, this example demanded much more processing time due to the number of arcs created by eulerization when many there are several paths. This confirms that eulerization takes much more time than test case generation since it is clearly visible through the number of arcs added (7860) in order to reach a full eulerized graph. On the other hand, Transition Tour was not that much affected by complexity of a system and could run faster than the previous case.

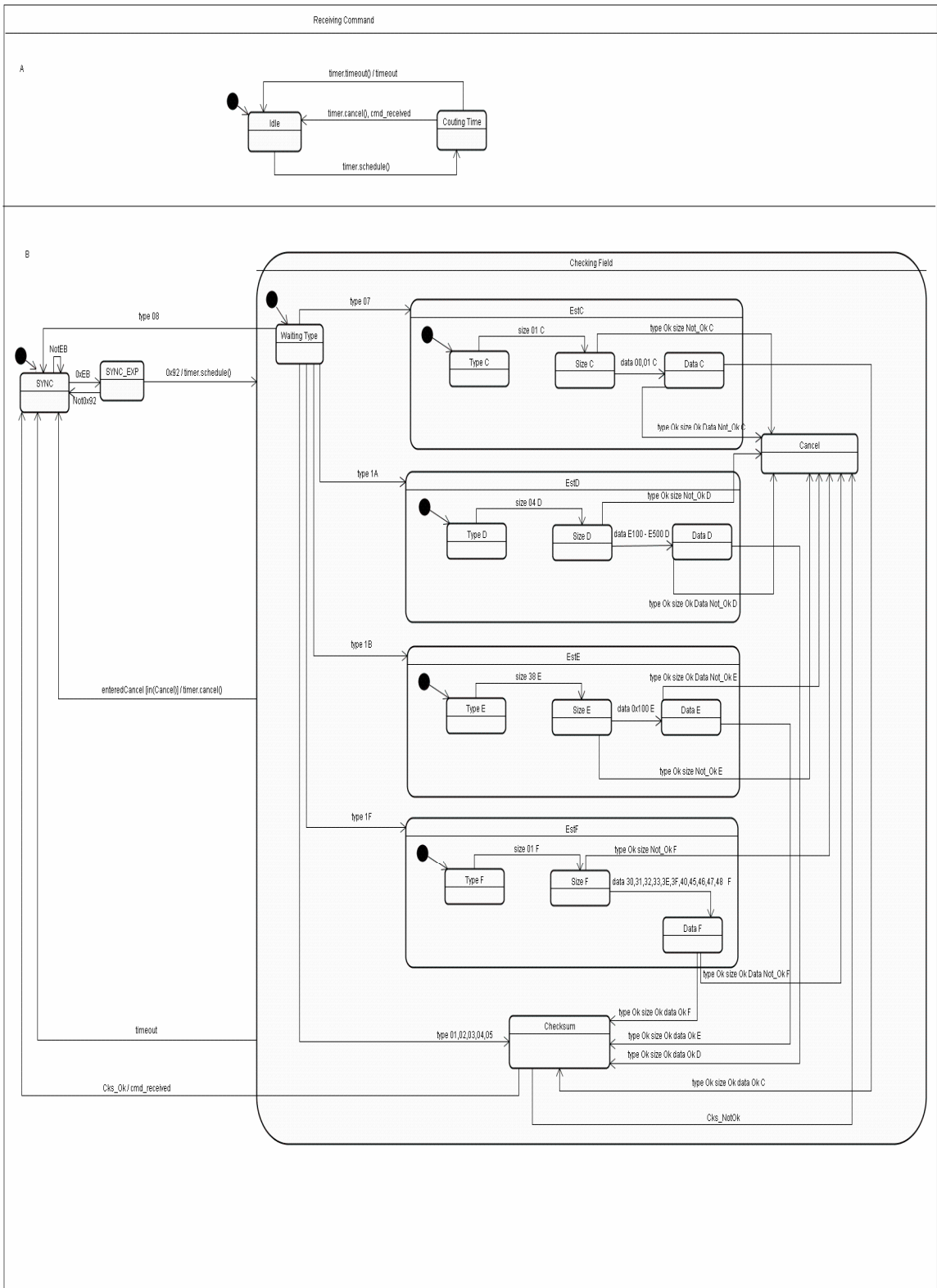


Figure 4.8 - Statecharts representation of APEX system detailed

In general, based on case studies, it is possible to deduce that two main discrepancies were observed: the execution time in favor of Transition Tour and the effectiveness in favor of Switch Cover. As an example, fifth case study was chosen to be discussed since it is a real application and also simple to understand. Consider its respective test cases generated from Transition Tour (Table 4.12) and Switch Cover (Table 4.13) methods.

As can be seen analyzing Table 4.12, Table 4.13 and FSM in Figure 4.6, Transition Tour method, as expected, is faster in terms of performance, but less precise by stimulating 70 events in order to cover the full graph with its unique test case sequence. Switch Cover method covered all possible 7 paths by stimulating just 27 events. However, it spent much more time. Both methods must start and finish test sequences from an initial state (IdleWaitingSync, in this case).

Generation of outputs is an important resource for testing activities and it was implemented in PerformCharts. As could be seen in the fourth case study, this feature was kept in WEB-PerformCharts. It is also important to clarify that WEB-PerformCharts also maintains other features already implemented in PerformCharts: transition probabilities and entry by history. Tests were conducted to ensure that the original functionalities within PerformCharts continue working within WEB-PerformCharts.

Table 4.12 - Test cases from Transition Tour in fifth case study

Step	Event	State
1	NotEB9	IdleWaitingSync
2	EB9	CountingTimeWaitingExpid
3	ExpidRec	CountingTimeWaitingType
4	TypeRec	CountingTimeWaitingSize
5	SizeRec	CountingTimeWaitingData
6	DataRec	CountingTimeWaitingChecksum
7	ChecksumRec	IdleWaitingSync
8	NotEB9	IdleWaitingSync
9	EB9	CountingTimeWaitingExpid
10	WaitingTimeExpired	IdleWaitingSync
11	NotEB9	IdleWaitingSync
12	EB9	CountingTimeWaitingExpid
13	ExpidRec	CountingTimeWaitingType
14	WaitingTimeExpired	IdleWaitingSync
15	NotEB9	IdleWaitingSync
16	EB9	CountingTimeWaitingExpid
17	WaitingTimeExpired	IdleWaitingSync
18	NotEB9	IdleWaitingSync
19	EB9	CountingTimeWaitingExpid
20	ExpidRec	CountingTimeWaitingType
21	TypeRec	CountingTimeWaitingSize
22	WaitingTimeExpired	IdleWaitingSync
23	NotEB9	IdleWaitingSync
24	EB9	CountingTimeWaitingExpid
25	WaitingTimeExpired	IdleWaitingSync
26	NotEB9	IdleWaitingSync
27	EB9	CountingTimeWaitingExpid
28	ExpidRec	CountingTimeWaitingType
29	WaitingTimeExpired	IdleWaitingSync
30	NotEB9	IdleWaitingSync
31	EB9	CountingTimeWaitingExpid
32	WaitingTimeExpired	IdleWaitingSync
33	NotEB9	IdleWaitingSync
34	EB9	CountingTimeWaitingExpid
35	ExpidRec	CountingTimeWaitingType
36	TypeRec	CountingTimeWaitingSize
37	SizeRec	CountingTimeWaitingData
38	WaitingTimeExpired	IdleWaitingSync
39	NotEB9	IdleWaitingSync

Continua

Table 4.12 - Conclusão

40	EB9	CountingTimeWaitingExpid
41	WaitingTimeExpired	IdleWaitingSync
42	NotEB9	IdleWaitingSync
43	EB9	CountingTimeWaitingExpid
44	ExpidRec	CountingTimeWaitingType
45	WaitingTimeExpired	IdleWaitingSync
46	NotEB9	IdleWaitingSync
47	EB9	CountingTimeWaitingExpid
48	WaitingTimeExpired	IdleWaitingSync
49	NotEB9	IdleWaitingSync
50	EB9	CountingTimeWaitingExpid
51	ExpidRec	CountingTimeWaitingType
52	TypeRec	CountingTimeWaitingSize
53	WaitingTimeExpired	IdleWaitingSync
54	NotEB9	IdleWaitingSync
55	EB9	CountingTimeWaitingExpid
56	WaitingTimeExpired	IdleWaitingSync
57	NotEB9	IdleWaitingSync
58	EB9	CountingTimeWaitingExpid
59	ExpidRec	CountingTimeWaitingType
60	WaitingTimeExpired	IdleWaitingSync
61	NotEB9	IdleWaitingSync
62	EB9	CountingTimeWaitingExpid
63	WaitingTimeExpired	IdleWaitingSync
64	NotEB9	IdleWaitingSync
65	EB9	CountingTimeWaitingExpid
66	ExpidRec	CountingTimeWaitingType
67	TypeRec	CountingTimeWaitingSize
68	SizeRec	CountingTimeWaitingData
69	DataRec	CountingTimeWaitingChecksum
70	WaitingTimeExpired	IdleWaitingSync

Table 4.13 - Test cases from Switch Cover in fifth case study

Step	Event	State
1	EB9	CountingTimeWaitingExpid
2	WaitingTimeExpired	IdleWaitingSync
-	-	-
3	EB9	CountingTimeWaitingExpid
4	ExpidRec	CountingTimeWaitingType
5	WaitingTimeExpired	IdleWaitingSync
-	-	-
6	EB9	CountingTimeWaitingExpid
7	ExpidRec	CountingTimeWaitingType
8	TypeRec	CountingTimeWaitingSize
9	WaitingTimeExpired	IdleWaitingSync
-	-	-
10	EB9	CountingTimeWaitingExpid
11	ExpidRec	CountingTimeWaitingType
12	TypeRec	CountingTimeWaitingSize
13	SizeRec	CountingTimeWaitingData
14	WaitingTimeExpired	IdleWaitingSync
-	-	-
15	EB9	CountingTimeWaitingExpid
16	ExpidRec	CountingTimeWaitingType
17	TypeRec	CountingTimeWaitingSize
18	SizeRec	CountingTimeWaitingData
19	DataRec	CountingTimeWaitingChecksum
20	WaitingTimeExpired	IdleWaitingSync
-	-	-
21	EB9	CountingTimeWaitingExpid
22	ExpidRec	CountingTimeWaitingType
23	TypeRec	CountingTimeWaitingSize
24	SizeRec	CountingTimeWaitingData
25	DataRec	CountingTimeWaitingChecksum
26	ChecksumRec	IdleWaitingSync
-	-	-
27	NotEB9	IdleWaitingSync
-	-	-



## 5 CONCLUSION

Decentralized work is a very common trend for widely dispersed companies in modern days, since it can result in time and cost savings decreasing travel and infrastructure requirements, instead of maintaining huge, centralized and expensive buildings. Based on this, WEB-PerformCharts was idealized with the decision for using an on-line database as storage method allowing test designers to share their projects, and facilitating the control of versions since its management is easier than copying multiple local files from multiple computers. Another relevant fact is that WEB-PerformCharts has other advantages when compared to conventional local systems since it can be accessed from any place in the world at anytime with a computer or laptop, an internet connection and a web browser enabling support to test process in a distributed environment.

In order to generate test cases remotely and independently from any other tool, Transition Tour and Switch Cover methods were implemented and integrated with WEB-PerformCharts. The task of incorporating more than one test sequence generation methods in WEB-PerformCharts can contribute to enable an efficiency comparison of different methods, and a brief comparison of the two methods was shown generating test cases for problems with varied complexity. Besides Transition Tour and Switch Cover, other methods can be implemented and integrated as cartridges of the system. In addition, the use of XML formatted documents represents an important step bringing another major contribution in standardization of test data. In future, studies will be made for integration between WEB-PerformCharts and tools that perform automatic test execution in order to improve the automation of test process activities.

As a secondary focus, it could be concluded that complex software modeling requires features as explicit representation of hierarchy and parallel activities. Therefore, a higher-level technique based on state-transitions diagrams is recommended. In this respect, Statecharts come into picture. However, dealing with higher-level techniques increases complexity in developing an automated

environment and demands more computational effort and one must be prepared to pay the price.

Depending on the number of states and arcs of the generated FSM, common sense says that the problem can be intractable. Fortunately it does not happen with case studies in this work when using WEB-PerformCharts methods.

In this work, both the tested methods were capable of generating test sequences from several case studies of different complexities and, the main observed discrepancy is the time to generate test cases for complex specifications. In terms of performance, while Transition Tour method reached good results covering full graph in a less time, Switch Cover takes a longer time in order to perform its more complex and precise algorithm. Therefore, small graphs can be easily processed by any method quickly; but for complex graphs, a method must be carefully chosen. Tester has to opt between waiting more time for a complete set of sequences from Switch Cover or obtain a fast unique set of sequences from Transition Tour.

It is also important to mention that, from the research conducted in this work, there are many other related works and several future studies that can complement with better results in real applications for space researches.

Studies for implementation and integration of new methods as D-Method and UIO-Method are being conducted already. The use of coverage criteria is an interesting possibility in order to treat complex specifications. A graphical interface for elaboration of Statecharts diagrams is under development and, when concluded, will be carefully studied to be adapted for WEB-PerformCharts tool. A useful study is to improve security for WEB-PerformCharts, maybe with cryptography or electronic signatures. Another study is related to the integration of WEB-PerformCharts directly with the tool that does automatic execution of test cases, as well as the minimization of FSM and test cases, since redundancies can exist for complex specifications. Finally, in this work test sequences were compared in terms of processing time to be generated and

graph coverage, but to determine which is a better method is not quite simple based only on this information without evaluating the quality of sequences. One more interesting study that can be conducted, based on these test cases, refers to the quality of sequences. Mutant analysis (FABBRI et al., 1999) is an error-based criteria usually applied for software testing since it is used to evaluate the quality of test case sets. It basically consists of generating mutant programs based on a set of mutation operators which are themselves based on common typical errors committed by programmers. Of course this evaluation is a complex process and it is not in the scope of this dissertation; however, such evaluation is being explored in other research under development.



## REFERENCES

AMARAL, A. S. M. S.; VELOSO, R. R.; VIJAYKUMAR, N. L.; FRANCÊS, C. R. L.; OLIVEIRA, E. On proposing a markup language for statecharts to be used in performance evaluation. **International Journal of Computational Intelligence**, v. 1, n. 3, p. 260-265, 2004.

AMARAL, A. S. M. S. **Geração de casos de teste para sistemas especificados em statecharts**. 2005. 162 p. (INPE-14215-TDI/1116). Dissertação (Mestrado em Computação Aplicada) - Instituto Nacional de Pesquisas Espaciais, São José dos Campos. 2005. Disponível em: <<http://urlib.net/sid.inpe.br/MTC-m13@80/2006/02.14.19.24>>. Acesso em: 06 out. 2008.

APFELBAUM, L.; DOYLE, J. Model Based Testing. In: INTERNATIONAL SOFTWARE QUALITY WEEK CONFERENCE, 10., 1997, San Francisco, USA. **Proceedings...** [S.l: s.n.], 1997.

ARANTES, A. O.; VIJAYKUMAR, N. L.; SANTIAGO, V. A.; CARVALHO, A.R. Automatic test case generation through a collaborative web application. In: IASTED INTERNATIONAL CONFERENCE INTERNET & MULTIMEDIA SYSTEMS & APPLICATIONS, 2008, Innsbruck, Austria. **Proceedings...** Calgary : IASTED-EuroIMS, 2008. p. 27-32.

BAFOUTSOU, G.; MENTZAS, G. A comparative analysis of web-based collaborative systems. In: INTERNATIONAL WORKSHOP ON DATABASE AND EXPERT SYSTEMS APPLICATIONS, 12., 2001, [S.I.]. **Proceedings...** [S.l: s.n.], 2001. p. 496-500.

BINDER, R. V. **Testing object-oriented systems: models, patterns and tools**. Boston, MA: Addison-Wealey, 2000.

CHEN, P. **The entity-relationship model: Toward a Unified View of Data**. Massachusetts Institute of Technology, Massachusetts: Cambridge, 1976.

CHOW, T. S. Testing Software Design Modeled by Finite-State Machines. **IEEE Transactions on Software Engineering**, v. 4, n. 3, p. 178-187, May 1978.

DAHURA, A. T.; SABNANI, K.; UYAR, M. U. Formal methods for generating protocol conformance test sequences. **Proceedings of the IEEE**, v. 70, n. 8, p. 1317-1326, August 1990.

DERDERIAN, K.; HIERONS, R. M.; HARMAN, M.; GUO, Q. Automated unique input output sequence generation for conformance testing of FSMs. **The Computer Journal**, v. 49, n. 3, p. 331-344, 2006.

ELLSBERGER, I.; HOGREFE, D.; SARMA, A. **SDL: Formal object-oriented language for communicating systems**. Prentice Hall Europe, 1997. ISBN(0-13-632886-5).

FABBRI, S. C. P. F.; MALDONADO, J. C.; SUGETA, T.; MASIERO, P. C. Mutation testing applied to validate specifications based on statecharts. In: INTERNATIONAL SYMPOSIUM ON SOFTWARE RELIABILITY ENGINEERING, 10., 1999, [S.l.]. **Proceedings...** [S.l: s.n.], 1999. p. 210.

GANE, C.; SARSON, T. **Structured systems analysis: tools and techniques**. Englewood Cliffs, N.J.: Prentice-Hall, 1979.

HAREL, D. **Statecharts: a visual formalism for complex systems**. Science of Computer Programming, v. 8, p. 237-274, 1987.

HAREL, D.; PNUELI, A.; SCHMIDT, J.; SHERMAN, R. On the formal semantics of statecharts. **IEEE Symposium on Logic in Computer Science**, Ithaca, USA, 1987.

HAREL, D.; NAAMAD, A. The statemate semantics of statecharts. **ACM Transactions on Software Engineering**, v. 5, n. 4, p. 293-333, 1996.

HAREL, D.; POLITI, M. **Modeling reactive systems with statecharts: the Statemate Approach**. USA: McGraw-Hill, 1998.

HARTMAN, A. **Model based test generation survey**. Agedis Consortium. Disponível em: <<http://www.agedis.de/>>. Acesso em: 30 out. 2007.

INPE, National Institute for Space Research. **EXP-OBDR communication protocol definition: a case study for PLAVIS**. São José dos Campos: INPE Internal Publication/QSEE Project, p. 9, 1998.

LEE, D.; YANNAKAKIS, M. Principles and methods of testing finite state machines – a survey. **Proceedings of the IEEE**, v. 84, n. 8, 1996.

MARTINS, E.; SABIÃO, S. B.; AMBRÓSIO, A. M. Condata: a tool for automating specification-based test case generation for COMMUNICATION SYSTEMS. IN: HAWAII International Conference on System Sciences (HICSS'33), 33., 2000, Maui, USA. **Proceedings...** [S.l: s.n.], 2000.

MATHWORKS. **Stateflow**. Disponível em: <<http://www.mathworks.com/products/stateflow>>. Acesso em: 11 jul. 2008.

MATTIELLO-FRANCISCO, M. F.; SANTIAGO JÚNIOR, V. A.; AMBRÓSIO, A. M.; COSTA, R.; LEISE, J. Verificação e validação na terceirização de software embarcado em aplicações espaciais. In: SIMPÓSIO BRASILEIRO DE QUALIDADE DE SOFTWARE, 2006. **Proceedings...** 2006. Papel. (INPE-14073-PRE/9242). Disponível em: <<http://urlib.net/sid.inpe.br/mtc-m16@80/2006/08.21.20.22>>. Acesso em: 06 out. 2008.

MYERS, G. **The art of software testing**. John Wiley & Sons, 2004.

OMG. **UML 2.0 infrastructure specification**. Disponível em: <<http://www.omg.org/cgi-bin/doc?formal/05-07-05>>. Acesso em: 11 jul. 2008.

PETERSON, J. L. **Petri net theory and modeling of systems**. London: Prentice-Hall International, 1981.

PIMONT, S.; RAULT, J.C. An approach towards reliable software. In: International Conference on Software Engineering, 4., 1979, Munich, Germany. **Proceedings...** [S.l: s.n.], 1979. p. 220-230.

PRESSMAN, R. S. **Software engineering: a practitioner's approach**. 5th edition. McGraw-Hill International Editions, 2000.

ROBINSON, H. Graph theory techniques in model-based testing. In: INTERNATIONAL CONFERENCE TESTING COMPUTER SOFTWARE, 16., 1999, Washington, USA. **Proceedings...** [S.l: s.n.], 1999.

SANTIAGO, V.; AMARAL, A. S. M. S.; VIJAYKUMAR, N. L.; MATTIELLO-FRANCISCO, M. F.; MARTINS, E.; LOPES, O. C. A practical approach for automated test case generation using statecharts. In: INTERNATIONAL WORKSHOP ON TESTING AND QUALITY ASSURANCE FOR COMPONENT-BASED SYSTEMS, 2. (TQACBS 2006), 2006, Chicago. **Proceedings...** 2006. (INPE-14044--PRE/9218). Disponível em: <<http://urlib.net/sid.inpe.br/mtc-m16@80/2006/08.10.12.08>>. Acesso em: 06 out. 2008.

SANTIAGO, V.; MATTIELLO-FRANCISCO, F.; COSTA, R.; SILVA, W. P.; AMBROSIO, A. M. QSEE Project: an experience in outsourcing software development for space applications. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING (SEKE'07), 9., 2007, Boston. **Proceedings...** [S.l.]: KSY, 2007.

SANTIAGO, V.; VIJAYKUMAR, N. L.; GUIMARÃES, D.; AMARAL, A. S.; FERREIRA, E. An environment for automated test case generation from statechart-based and finite state machine-based behavioral models. In: INTERNATIONAL CONFERENCE ON SOFTWARE TESTING VERIFICATION AND VALIDATION (ICST 2008), 1., 2008, Lillehammer, Norway. **Proceedings...** [S.l: s.n.], 2008.

SIDHU, D.; LEUNG, T. Formal methods for protocol testing: a detailed study. **IEEE Transactions on Software Engineering**, v. 15, n. 4, p. 413-426, 1989.

SOMMERVILLE, I. **Software engineering**. 6.ed. [S.l]: Addison Wesley, 2003.

TIAN G. Y.; TAYLOR, D. Design and implementation of a web-based distributed collaborative design environment. In: International Conference on Information Visualisation, 5., 2001, London, UK. **Proceedings...** [S.l]: IEEE, 2001. p. 703-707.

VIJAYKUMAR, N. L.; CARVALHO, S. V.; ABDURAHIMAN, V. On proposing statecharts to specify performance models. **International Transactions in Operational Research**, v. 9, n. 3, p. 321-336, 2002.

W3C. **XML: extensible markup language**. Disponível em:  
<<http://www.w3.org/XML/Activity>>. Acesso em: 11 jul. 2008.

XIAO, H.; ZHIYI, M.; WEIZHONG, S.; SHAO, G. A metamodel for the notation of graphical modeling languages. In: Computer Software and Applications Conference (COMPSAC), 31., 2007, [S.l.]. **Proceedings...** [S.l]: IEEE, 2007. v. 1, p. 219-224.



## **PUBLICAÇÕES TÉCNICO-CIENTÍFICAS EDITADAS PELO INPE**

### **Teses e Dissertações (TDI)**

Teses e Dissertações apresentadas nos Cursos de Pós-Graduação do INPE.

### **Manuais Técnicos (MAN)**

São publicações de caráter técnico que incluem normas, procedimentos, instruções e orientações.

### **Notas Técnico-Científicas (NTC)**

Incluem resultados preliminares de pesquisa, descrição de equipamentos, descrição e ou documentação de programa de computador, descrição de sistemas e experimentos, apresentação de testes, dados, atlas, e documentação de projetos de engenharia.

### **Relatórios de Pesquisa (RPQ)**

Reportam resultados ou progressos de pesquisas tanto de natureza técnica quanto científica, cujo nível seja compatível com o de uma publicação em periódico nacional ou internacional.

### **Propostas e Relatórios de Projetos (PRP)**

São propostas de projetos técnico-científicos e relatórios de acompanhamento de projetos, atividades e convênios.

### **Publicações Didáticas (PUD)**

Incluem apostilas, notas de aula e manuais didáticos.

### **Publicações Seriadas**

São os seriados técnico-científicos: boletins, periódicos, anuários e anais de eventos (simpósios e congressos). Constam destas publicações o Internacional Standard Serial Number (ISSN), que é um código único e definitivo para identificação de títulos de seriados.

### **Programas de Computador (PDC)**

São a seqüência de instruções ou códigos, expressos em uma linguagem de programação compilada ou interpretada, a ser executada por um computador para alcançar um determinado objetivo. São aceitos tanto programas fonte quanto executáveis.

### **Pré-publicações (PRE)**

Todos os artigos publicados em periódicos, anais e como capítulos de livros.