A Practical Approach for Automated Test Case Generation using Statecharts*

Valdivino Santiago[1], Ana Silvia Martins do Amaral[1], N. L. Vijaykumar[1], Maria de Fátima Mattiello-Francisco[1], Eliane Martins[2] and Odnei Cuesta Lopes[3]
[1]National Institute for Space Reseach, [2]University of Campinas, [3]DBA Engenharia de Sistemas
valdivino@das.inpe.br, {anasil, vijay}@lac.inpe.br, fatima@dss.inpe.br, eliane@ic.unicamp.br, olopes@dba.com.br

Abstract

This paper presents an approach for automated test case generation using a software specification modeled in Statecharts. The steps defined in such approach involve: translation of Statecharts modeling into an XML-based language; and the PerformCharts tool generates FSMs based on control flow. These FSMs are the inputs for the Condado tool which generates test cases. The idea is to demonstrate that by using a higher-level technique, such as Statecharts, complex software can be represented with clarity and rich details. A case study was on an implementation of a protocol specified for communication between a scientific experiment and the On-Board Data Handling Computer of a satellite under development at National Institute for Space Research (INPE).

1. Introduction

Software for satellite's on-board computers is reactive by nature, responding to stimuli known as events. Such software is complex due its close interaction with the computer hardware, sensors, actuators and other devices present in the satellite, and hard to be replaced in case of faults due to the unmanned aspect of the mission. Due to this fact, verification and validation play an important role during the whole software life cycle.

Wrong interpretations of complex software from non-formal specification can result in incorrect implementations leading to testing them for conformance to its specification standard [1]. Besides, tests by introducing errors into input data, for example, are particularly important in order to see its behavior in the presence of non-specified events. In order to conduct both verification and validation or even to determine its performance, the software shall be represented formally to be handled through a computer. Finite State Machines (FSMs) [2] are popular, formal and are a natural choice to represent reactive systems. They consist of states (vertices) moving to other states by means of transition arcs (edges) triggered by events and are depicted through state transition diagrams. Several methods [1] and [3] have been proposed to generate test sequences from a FSM representation. In a similar way, if a reactive system is represented as a FSM and shown that it is a Markov chain, one can use an analytical approach to obtain performance measures [4].

Complex software usually deals with parallel activities. Representation of encapsulation would be an additional advantage. FSMs cannot explicitly represent such features, so, this leads to considering higher-level techniques. However, these techniques must be formal so that they can be computationally handled. At this stage, one has to bear in mind that a price has to be paid in order to deal with such specifications, as there might not be a straightforward solution to handle test sequences or performance evaluation. This means that this matter has to be evaluated by the user before taking a decision whether to use FSM or move to a higher-level technique. It is expected that the decision, in most cases, will depend on the complexity of the system to be verified or validated.

The focus of this paper is towards test sequence generation from a specification of a reactive system, a space application software, in Statecharts [5] and the use of PerformCharts [6]. In order to adapt PerformCharts to generate test sequences, it has been associated to a test case generation method switch cover implemented within the Condado tool [7]. A case study on a real satellite's scientific experiment is explored to explain the overall process from

* This work was supported in part by Financiadora de Estudos e Projetos (FINEP) through Qualidade do Software Embarcado em Aplicações Espaciais (QSEE) project.

specification to test sequence generation. The paper is organized as follows. Section 2 briefly explains both PerformCharts and Condado. Section 3 discusses how these tools have been integrated in order to deal with test sequence generation. Section 4 addresses the case study of a protocol implementation used for communicating a scientific experiment and the On-Board Data Handling Computer (OBDH) of a Brazilian scientific satellite. Section 5 shows some results while section 6 concludes the paper.

2. PerformCharts and Condado

Statecharts extend state-transition diagrams with notions of hierarchy (depth), orthogonality (parallel activities) and interdependence/synchronization (broadcast communication) [5] and [8]. Statecharts consist of states, conditions, events, actions and transitions. Their subset was used in performance evaluation [6]. Events are interpreted as: they can be: internal (or immediate) and triggered automatically (not explicitly stimulated) taking zero time when enabled; or external events that are stochastic (follow an exponential distribution) and are explicitly stimulated. Statecharts specification is converted into a Markov chain by: from an initial configuration (basic states of each orthogonal component in the initial instant), enabled immediate events are triggered and new configurations are obtained; for these new configurations, enabled stochastic events are stimulated yielding new configurations; once a configuration is obtained, internal events, if enabled, are triggered, firing transitions to yield new configurations; this stimulation goes on until all the configurations have been expanded. The result is a set where each element consists of a source configuration, stimulated stochastic event (along with its transition rate with exponential distribution), and the target configuration. This set is a Markov chain and when solved steady-state probabilities are obtained [4]. Figure 1(a) shows an example with three parallel components that correspond to two machines (E1 and E2) and a supervisor (Supervisor) to repair any eventual failure of the machines. The corresponding Markov chain is shown in Figure 1(b).

Note that the number of configurations obtained (10) is less than the product (27) of the three machines in Fig.1. Details of the application of Statecharts in performance models can be seen in [6].

Statecharts specification is written in XML-bassed language PerformCharts Markup Language (PcML) [7] and scripts developed in Perl interpret PcML and generate a main program in C++ language that creates the necessary data structures to hold the specification and calls to methods to produce the Markov chain and its solution (steady-state probabilities).

Condado is a test case generation tool for FSM. Condado stems from “control” and “data”. The control aspect is related to the valid sequences of events that the machine represents. The data aspect allows parameters of events to be generated. For the control part, Condado implements the switch cover method [3]. A switch is a transition-to-transition pair. The method generates test cases to cover all pairs of transitions in the model. State transition diagrams are directed graphs. Therefore, algorithms from graph theory can be used in order to traverse automatically.

The algorithm implemented in Condado is known as sequence of “de Bruijn”. The FSM of Figure 2(a) will help to illustrate the algorithm whose steps are:

Step 1 – A dual graph is created from the original one, by converting arcs into nodes;

Step 2 – By considering 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 2(b));

Step 3 – The dual graph is transformed into a “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. This “Eulerized” graph that corresponds to Figure 2(b) is shown in Figure 2 (c);

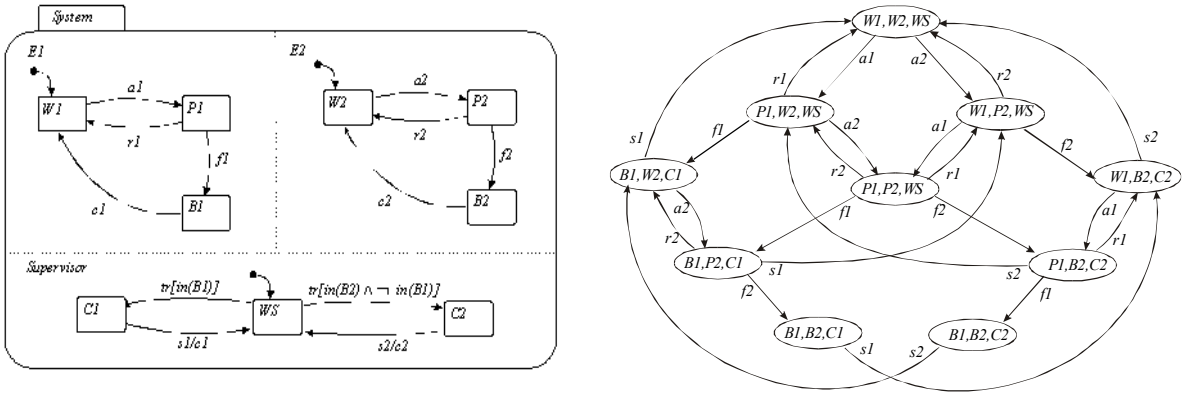
Step 4 – Finally, the nodes are traversed registering those that are visited, generating the following test sequences: abf, abrbf, cf, crbf.

In order to use Condado, the FSM must be: a Mealy machine; initially connected; no need to be complete; and minimum, that is, does not contain redundant states.

In order to keep the number of test cases manageable, the number of states and inputs must be kept small; otherwise, the number of test cases may explode. In the next section a method is presented that allows a reduced machine to be used as input to Condado.

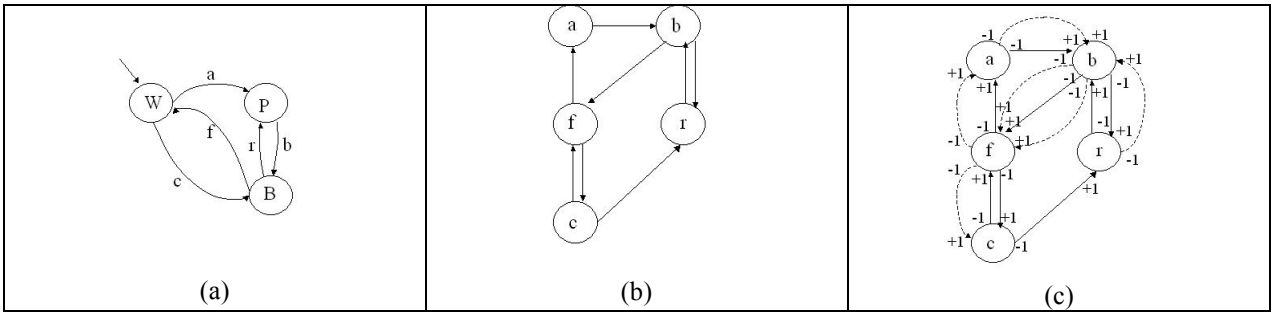
3. Integrating PerformCharts and Condado

In order to use PerformCharts for test sequence generation, some additional elements from Statecharts such as Variables and Expressions have been included. PerformCharts is used in the same fashion as it is used for performance evaluation.



(a) (b)

Figure 1. Statecharts representation. (a) System with two machines and a repairer; (b) Resulting Markov Chain



(a) (b) (c)

Figure 2. Condado algorithm. (a) FSM example; (b) Dual graph with arcs; (c) "*Eulerized*" Graph.

However, the output is not a Markov chain (and the steady-state probabilities) but a state-transition diagram. The state-transition diagram, after being converted into a Prolog base of facts, is then fed to Condado tool so that test cases are generated to be executed at a later stage [7]. The FSM obtained from PerformCharts is provided in XML. Then an XSLT parser was used to generate the required base of facts. In a nutshell, the methodology to convert Statecharts specification of a software into test cases can be described in the following steps: (a)the system specification must be written in PcML, presented in section 2; (b)by using a Perl script, the PcML specification is converted into a text file containing the C++ main program; (c)a scenario is created by assigning values to the different fields of a communication protocol frame. Each scenario has a unique FSM associated which is generated through the PerformCharts tool. As mentioned earlier, the Condado tool implements the switch cover method to generate test cases. In order to avoid test case explosion a solution has been adopted which assigns a value of zero to a selected event. The effect is that the transition associated to that event is not executed thereby producing a smaller machine; (d)the FSM generated by the PerformCharts must be written in the input format required by Condado tool. The output of the converted state-transition diagram in XML format is translated by using XSLT into the input required by Condado; and (e)finally, Condado tool generates the test cases for the system initially specified in Statecharts.

4. Case Study

In order to show the integrated use of Condado and PerformCharts for test case generation, a protocol specification [9] developed for the communication between a scientific experiment and the OBDH of a Brazilian scientific satellite has been chosen. The Implementation Under Test (IUT) is the command recognition component of the on-board software for an astrophysical experiment to be included in the Equatorial Atmosphere Research Satellite (EQUARS).

Although the main focus of this paper is test case generation, tests created were executed in a simulated version of this experiment, developed in Java. The OBDH was simulated and executed in a different microcomputer. The communication is in master-slave mode, where the experiment is totally controlled by

OBDH. The command message sent by OBDH to the experiment is composed of 6 fields: SYNC (EB9 synchronization value), EID (experiment identification), TYPE (specifies accepted commands), SIZE (amount of Bytes in the DATA field), DATA and CKS (8-bit checksum). SIZE and DATA fields are optional and depend on the type of command.

The command recognition component behavior of the communication protocol is shown in Figure 3. Table 1 provides some mnemonics to help its understanding.

Referring to Figure 3, the initial configuration is (*Idle*, *Waiting Sync*). All fields of the command message are verified by the experiment through on-board software. For instance, suppose 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. In the B macro-state, the event EB9 changes its sub-state to *Waiting ExpId* and the action *starting timing counting* makes the A state change from sub-state *Idle* to sub-state *Counting Time*. After the EB9, the *eid rc[eid = 2]* event is triggered because EID = 2. The B state moves from *Waiting ExpId* 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 state switches from *Waiting Checksum* to *Waiting Sync* in order to wait for another command. Also, the action *command received* means the message was received and accepted by the experiment software. It also changes the A state from *Counting Time* to *Idle*. With this action, the timing counting is interrupted and the software is able 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 from OBDH. No message on problems in receiving a command is reported back to OBDH.

A timeout mechanism is implemented in both the computers depicted in state A. For example, if the time defined for receiving a whole command from OBDH expires, the *waiting time expired [not in (Aborting)]* event is triggered in state A and the action/event *timeout* will change the state B from *Checking Field* to *Waiting Sync*.

Table 1 shows the values that can be assumed by the TYPE field (type and auxiliary variable tp) and those assumed by the DATA field (data). Also, from Table 1 and Figure 3, it can be noticed that TYPE values 01, 02, 03, 04 and 05 do not have the optional fields SIZE and DATA in the command message. On the other hand, TYPE values 07, 1A, 1B and 1F have such fields (states *Waiting Size* and *Waiting Data* in state B).

Table 1. Mnemonics in Figure 3.

Mnemonics	Meaning
EB9	Synchronism value
rc	Received. Indicates an event has happened
eid	Experiment identification
tp, sz	Auxiliary variables
cksum	Checksum
x	x = a or b or c or d
a	tp = 07 and (data = 00 or data = 01)
b	tp = 1A and (initial_address <= final_address)
c	tp = 1B and (address >= loading pointer)
d	tp = 1F and (data = 30 or data = 31 or data = 32 or data = 33 or data = 3E or data = 3F or data = 40 or data = 45 or data = 46 or data = 47 or data = 48)
y	type = 01 or type = 02 or type = 03 or type = 04 or type = 05 or type 07 or type = 1A or type = 1B or type = 1F

5. Results

According to step (a) of the methodology, the software specification was translated to PcML. A sample of the PcML specification follows: <State Name="A" Type="OR" Default="Idle">. Then, PcML was translated into a text file containing the C++ main program, whose sample is: ExpSci.createSonState ("A",OR,"ReceivingCommand"); (step (b)). A different FSM for each scenario (columns Sc) of Table 2 was generated (step (c)). A scenario was created by assigning values to the command message fields.

These scenarios shall be created in order to deal with the software behavior that is in conformance with the protocol specification as well as for situations where errors are introduced. For example, scenario 8 in Table 2 was created by assigning the following values: SYNC = EB9, EID = 2, TYPE = 1F, SIZE = 01, DATA = 32, CKS = 31 (correct value). This is a normal scenario which deals with conformance aspect of the software. Figure 4 shows the FSM generated for scenario 8.

On the other hand, scenario 4 was created with TYPE = 08. This is a non-specified value for the TYPE field so this scenario is related to the introduction of errors in the command fields. The idea of this paper was not to generate different FSMs for each different combination of the inputs but it was mainly to show how the methodology works. Table 2 shows test cases for all 9 scenarios resulting in 38 test cases.

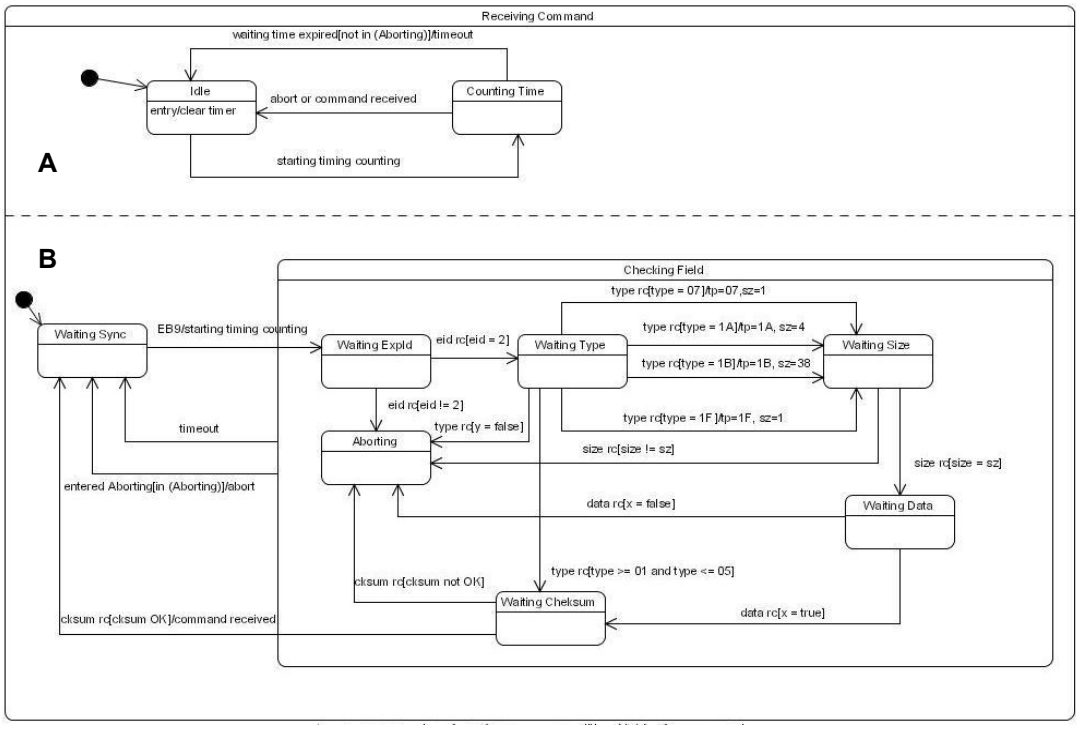


Figure 3. Statecharts modeling of the command recognition part of the communication protocol.

In Table 2, InSc means the amount of scenarios a TC belongs to. However, it is important to notice that as each scenario is a different FSM, some test cases are repeated in more than one scenario. This repetition can be explained by the fact some events occur according to the software specification and also due to errors regardless of a particular scenario. For example, consider test case A, in which event EB9 (B state) and event *waiting timing expired[not in (Aborting)]* (A state) occur. This test case is generated in all 9 scenarios (Table 2). Event EB9 is the first one expected by the software in a entire command message and a timeout can occur in any transaction between the computers. Test cases from A to J are those created by assigning values, not defined in the protocol specification, to the command message fields and also by not transmitting the whole command message which results in a timeout in the implementation under test. Test cases K and L are those with input values as defined in the specification.

A total number of 12 test cases generated were unique which implies in 31.6% of the total initially created. Although this number can indicate an excessive repetition of test cases, it is important to point out that there was not any excessive number of test cases generated. One could have used just the Condado tool by itself. In this case, the specification must be provided as a FSM. This is not an issue as such if the software is not too complex. Moreover, explosion of test cases frequently takes place and many of the test cases have a sequence of transitions that do not occur in the real implemented software. For instance, when the TYPE field is 1F, the protocol specifies that the SIZE must be 01 and DATA must be one of the options shown in event d (one of the component events of x) in Table 1. Condado combines all the transitions to generate test cases and a large number of test cases are generated.

Table 2. Scenarios (Sc) X Test Cases (TC)

TC	Sc 1	Sc 2	Sc 3	Sc 4	Sc 5	Sc 6	Sc 7	Sc 8	Sc 9	InSc
A	x	x	x	x	x	x	x	x	x	9
B	x	x	x	x	x	x	x	x	x	9
C	x	x	x		x	x		x	x	7
D	x	x						x		3
E	x							x		2
F	x									1
G		x								1
H			x							1
I				x						1
J					x					1
K						x			x	2
L								x		1
Total	6	5	4	3	4	4	2	6	4	38

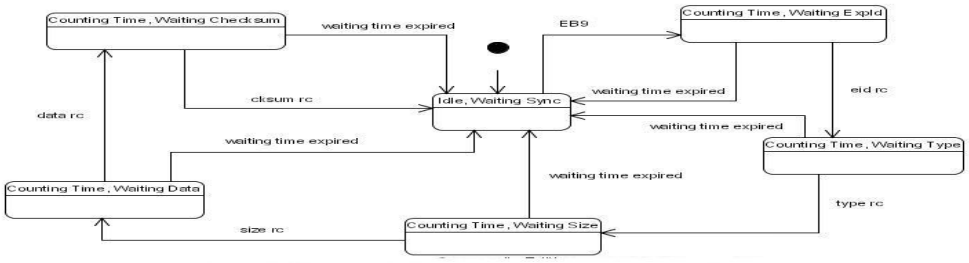


Figure 4 . FSM Generated for Scenario 8.

Test case explosion can be avoided by a careful selection of what arcs to be pruned which requires some effort by the tester. PerformCharts, while converting Statecharts specification into a FSM, does not generate a blind "AND" product of the basic states within each parallel component. The generated machine is in fact the possible combination of configurations based on the events that are stimulated. Moreover, one or more arcs can be pruned to avoid generating a larger machine. One has to note that pruning of arcs has a serious drawback of not testing the entire machine. On the other hand, applications of test case generating methods are feasible on complex systems.

In terms of execution, two errors were found in the Java version of the real experiment software. The first one occurred when test case A was executed. After receiving an EB9, the software must start counting the time. In test case A, no additional information is transmitted from OBDH after EB9. So the software implementation must indicate the occurrence of a timeout (waiting timing expired[not in (Aborting)] event triggered) but this did not happen. The second one is that, on purpose, a mutant version of the software was created so that it did not assume TYPE = 01 as a valid specified value. Only one mutant with a single fault was created at this time. Test case K in scenario 9 was executed and detected this error.

6. Conclusions

Complex reactive systems are one of a kind where many intricacies have to be represented. Statecharts, originally created for representing real-time systems, have been adopted to represent and deal (analytically) with performance models resulting into PerformCharts.

This paper presented an approach for automated test case generation using Statecharts as a modeling technique. Statecharts can provide hierarchy and parallelism and enable to model a complex system more realistically. The methodology described involve the use of PerformCharts and Condado to generate test sequences. The approach was applied on a simulated version of a satellite experiment software. The results were satisfactory. None of the test cases generated were meaningless. Moreover, there was no test case explosion. Although these conditions are not enough in order to guarantee a test case generation approach is successful they show a real improvement when compared with just the use of the Condado as a standalone tool with a FSM specification.

Implementation of a filter tool is required to identify and eliminate the repeated test cases in more than one scenario More scenarios will be derived and test cases will be executed to the IUT. Also, this approach will be applied in the entire software and not only in one of its components. The entire software of the experiment resulted in 13 pages of dynamic behavior modeling in Statecharts.

7. References

[1] Sidhu, D.; Leung, T. Formal Methods for Protocol Testing: A Detailed Study. *IEEE Transactions on Software Engineering*, 15(4), 413-426, 1989.

[2] Lee, D.; Yannakakis, M. Principles and methods of testing finite state machines – A survey. *Proceedings of the IEEE,* vol. 84(8), 1996, pp. 1090-1123.

[3] Pimont, S.; Rault, J.C. An approach towards reliable software. In *Proceedings of the 4th International Conference on Software Engineering*, Munich, Germany, pp. 220-230, 1979.

[4] Silva, E. A. S.; Muntz, K. M. Computational Methods to solve Markov Chains: Applications in Computing and Communication Systems. Instituto de Informática da UFRGS, Gramado, Brasil, p.195, 1992. (In Portuguese).

[5] Harel, D.; Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, v. 8, n. , p. 231-274, 1987.

[6] Vijaykumar, N. L.; Carvalho, S. V.; Abdurahiman, V. On proposing Statecharts to specify performance models. *International Transactions in Operational Research*, 9, 321-336, 2002.

[7] Amaral, A.S.M.S. Test case generation of systems specified in Statecharts. M.S. thesis – Laboratory of Computing and Applied Mathematics, INPE, Brazil, 2006. (In Portuguese).

[8] Harel, D.; Politi, M. Modeling reactive systems with Statecharts: the statemate approach. USA: McGraw-Hill, 1998.

[9] National Institute for Space Research (INPE). EXP-OBDH Communication Protocol Definition: a case study for PLAVIS. São José dos Campos: INPE/QSEE Project, 1998, p. 9 (INPE Internal Publication/QSEE Project)