Ministério da
Ciência e Tecnologia

INPE-16677-RPQ/850

MODEL-BASED TEST CASE GENERATION USING STATECHARTS AND Z: A COMPARISON AND A COMBINED APPROACH

Valdivino Alexandre de Santiago Júnior
Maximiliano Cristiá
Nandamudi Lankapalli Vijaykumar

Original document registry:
<http://urlib.net/sid.inpe.br/mtc-m19@80/2010/02.26.14.05>

INPE
São José dos Campos
2010

PUBLISHED BY:

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

Ministério da
Ciência e Tecnologia

INPE-16677-RPQ/850

MODEL-BASED TEST CASE GENERATION USING STATECHARTS AND Z: A COMPARISON AND A COMBINED APPROACH

Valdivino Alexandre de Santiago Júnior
Maximiliano Cristiá
Nandamudi Lankapalli Vijaykumar

Original document registry:
<http://urlib.net/sid.inpe.br/mtc-m19@80/2010/02.26.14.05>

INPE
São José dos Campos
2010

ABSTRACT

At *Instituto Nacional de Pesquisas Espaciais* (INPE - National Institute for Space Research), researchers and Software Engineers have been using Statechart-based testing for some time to test satellite computer embedded software. On the other hand, a group of researchers at *Centro Internacional Franco Argentino de Ciencias de la Información y de Sistemas* (CIFASIS - French Argentine International Center for Information Systems and Sciences) and Flowgate Consulting have been applying Z-based testing for unit testing. Both groups started to compare their approaches and tools and, what started as a comparison to share ideas and results, is now turning into the realization that actually both techniques complement and benefit from each other, yielding a more effective and wider Model-Based Testing (MBT) approach. This Technical Report details the ideas of the comparison between Statecharts and Z, and also the proposal combining these two techniques shown previously in a paper accepted and presented at the *11th IEEE Latin American Test Workshop* (LATW'10) that took place in Punta del Este, Uruguay, 2010.

GERAÇÃO DE CASOS DE TESTE BASEADA EM MODELOS □ STATECHARTS E LINGUAGEM Z: UMA COMPARAÇÃO E □ UMA ABORDAGEM COMBINADA

RESUMO

No Instituto Nacional de Pesquisas Espaciais (INPE), pesquisadores e Engenheiros de Software têm usado testes baseados em Statecharts já há algum tempo para testar software embarcado em computadores de satélites. Por outro lado, um grupo de pesquisadores do *Centro Internacional Franco Argentino de Ciencias de la Información y de Sistemas* (CIFASIS) e *Flowgate Consulting* têm aplicado testes basedos em linguagem Z no escopo de testes de unidade. Ambos os grupos iniciaram uma comparação das respectivas abordagens e ferramentas e, o que se iniciou como uma comparação para compartilhar idéias e resultados, resultou em uma proposta de Testes Baseados em Modelos (TBM) mais ampla e efetiva, dado que se percebeu que ambas as técnicas se complementam e se beneficiam uma da outra. Este Relatório Técnico detalha as idéias da comparação entre Statecharts e Z, e também a proposta combinando estas duas técnicas mostradas anteriormente em um artigo aceito e apresentado no *11th IEEE Latin American Test Workshop* (LATW'10), o qual ocorreu em Punta del Este, Uruguai, em 2010.

CONTENTS

Pág.

LIST OF FIGURES

Pág.

LIST OF TABLES

Pág.

1 INTRODUCTION

Satellite software is a mission critical, reactive and data handler program that must be validated and verified thoroughly. In these cases, verification and validation always comprises testing although other techniques can be applied too. Testing a program which presents a reactive behavior and data handling functions as well, is a complex task that does not seem to be extensively addressed in an industrial setting. Furthermore, if it is desirable Model-Based Testing (MBT) to test both units and the complete system with tool support, then the informed results rapidly approaches to zero. In (NETO et al., 2008) and (HIERONS et al., 2009) such issues are not addressed, although (HIERONS et al., 2001) deals with them. Hence, in this Technical Report, the problem outlined above is addressed, in the context of space application software. In fact, this Technical Report details the ideas of the comparison between Statecharts and Z, and also the proposal combining these two techniques shown previously in a paper accepted and presented at the *11th IEEE Latin American Test Workshop* (LATW'10) that took place in Punta del Este, Uruguay, 2010.

This work is the outcome of what started as a comparison between two MBT approaches, one adopted at the *Instituto Nacional de Pesquisas Espaciais* (INPE - National Institute for Space Research) and another at the *Centro Internacional Franco Argentino de Ciencias de la Información y de Sistemas* (CIFASIS - French Argentine International Center for Information Systems and Sciences), and later became a combination of both techniques. The joint project started with the intention to see how test designers at INPE could improve the testing of satellite computer embedded software. At the beginning only one thing was clear: both groups were working on MBT but with different notations and tools. Then, both groups demonstrated their respective methods to each other using the same real-world case study proposed by INPE. The first conclusion was that both methods were functionally testing the same system but in different ways. Latter Paradkar's work (PARADKAR, 2005) was extended in order to accomplish the comparison between the approaches. Finally, it was realized that both approaches could be combined into a single one addessing model-based test case generation with the main benefits of the techniques: Statecharts to model behavior and Z to model the data space.

This Technical Report is organized as follows. Chapters 2 and 3 briefly introduce the approach used by each group. Chapter 4 details the case studies and the results of comparing both methods. Chapter 5 presents the proposal for combining Statechart-

based and Z-based testing. The conclusions are in Chapter 6.

2 STATECHARTS AND FINITE STATE MACHINES FOR TESTING AT INPE

INPE has been using MBT within research projects for system and acceptance testing. Statecharts (SANTIAGO et al., 2006; SANTIAGO et al., 2008) and Finite State Machines (FSMs) (AMBROSIO et al., 2007) are the main techniques used for modeling the behavior of the Implementation Under Test (IUT) for testing purposes. The application domain is software embedded into on-board computers of scientific satellites and balloons under development at INPE. These MBT approaches have shown efficiency in detecting defects in the source code.

One of the main advantages of FSMs is simplicity. Reactive systems, protocol implementations, classes of Object-Oriented applications are some examples commonly addressed by FSM modeling. Several test criteria (methods) may be used for model-based test case generation regarding the FSM technique. A few of such methods are the Transition Tour (TT), Distinguishing Sequence (DS), Unique Input/Output (UIO), W (SIDHU; LEUNG, 1989), switch cover (1-switch) (PIMONT; RAULT, 1976) and state counting (PETRENKO; YEVTUSHENKO, 2005). Despite their adoption by many researchers, FSMs are not too adequate for representing features such as parallelism and hierarchy. On the other hand, Statecharts provide a simple way to represent these characteristics. There are several approaches proposed to generate test cases from Statecharts models (BINDER, 1999) (SANTIAGO et al., 2006).

In system and acceptance testing, the entire software product is considered. If a professional wants to develop model-based test case generation relying on, for example, FSMs or Statecharts, he/she must develop the state-transition diagrams. Due to the state explosion problem, a test designer at INPE usually breaks down the entire system based on usage scenarios. Models are then derived to address each scenario and, provided they are small enough, test cases are generated from them.

INPE has been developing an environment named *Geração Automática de Casos de Teste Baseada em Statecharts* (GTSC - Automated Test Case Generation based on Statecharts) that allows test designers to model software behavior using Statecharts and/or FSMs in order to automatically generate test cases based on some test

criteria for FSM and some for Statecharts (SANTIAGO et al., 2008). At present, GTSC implements a version of switch cover, UIO and DS test criteria for FSM models and two test criteria from the *Statechart Coverage Criteria Family* (SCCF) (SOUZA, 2000) for Statecharts models: all-transitions and all-simple-paths. Figure 2.1 shows the GTSC architecture. Note that besides the architectural elements, it also shows external elements (Reachability Tree, Test Cases, ...) built during the use of GTSC by a test designer.

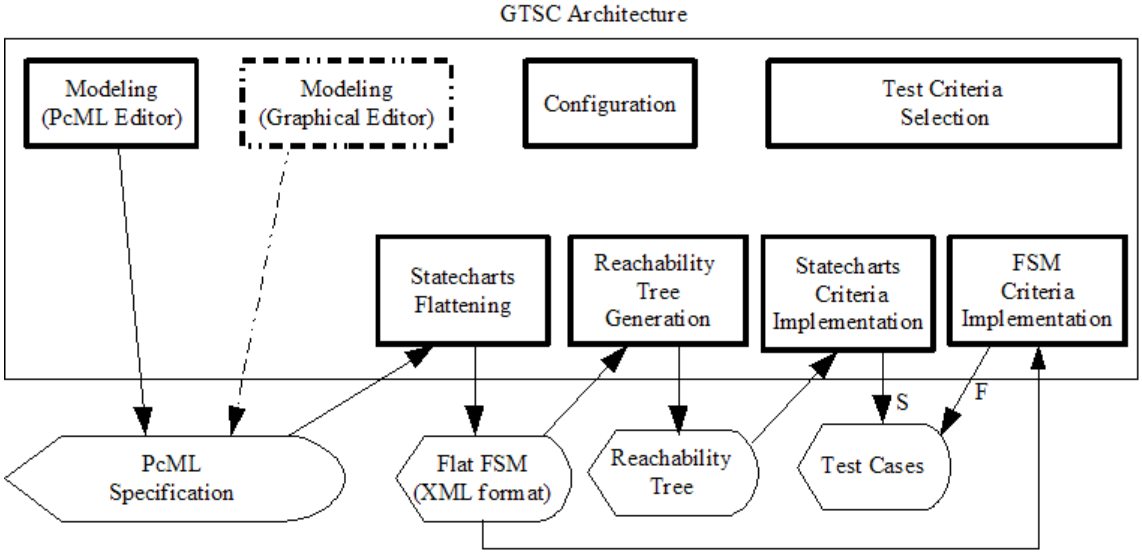

Figure 2.1 - The GTSC Architecture.

GTSC transforms a Statecharts model into a flat FSM, i.e. a model where all hierarchical and orthogonal features of the Statecharts were removed. Each state of the resulting flat FSM is actually a configuration of active BASIC states of the input model at a certain instant of time. This flat FSM is indeed the basis for test case generation. Hence, a test designer may follow two approaches:

a) If a SCCF test criterion will derive test cases, GTSC must adapt the flat FSM to resemble a reachability tree (MASIERO et al., 1994). Thus, based on the selected test criterion of SCCF and on this tree, test cases are created;

b) If an FSM test criterion is the option, it is only necessary for the user to choose among the available criteria for FSM and instruct the environment

to generate test cases, based on the flat FSM.

GTSC was able to generate flat FSMs with as many as 40 states (configurations) and more than 300 transitions, and test suites with up to 265 optimized test cases, showing its potential scalability for handling complex systems (SANTIAGO et al., 2008). INPE has also been developing a Web tool that allows model-based test case generation named WEB-Perform Charts (ARANTES et al., 2008). WEB-PerformCharts works similar to GTSC, by transforming a Statecharts model into a flat FSM, and it has implemented the TT, a version of switch cover and UIO test criteria for FSM models. Such a tool addresses collaborative work where different teams cooperate with an objective of reaching a specific goal, and it has generated flat FSMs as complex as the ones created by GTSC.

3 Z-BASED TESTING WITH FASTEST

Phil Stocks and David Carrington introduced in (STOCKS; CARRINGTON, 1996; STOCKS, 1994; MACCOLL; CARRINGTON, 1998) the Test Template Framework (TTF) to conduct MBT of Z specifications (SPIVEY, 1989). TTF includes a rigorous and disciplined technique for defining and structuring abstract test templates and cases[1]. They also proposed new testing tactics particularly well suited to the Z notation. Testing tactics are the mechanisms used to partition the input space into test templates and, in turn, test templates into more test templates, thus building a so called *testing tree*. Test cases are elements selected from the leaves of the testing tree.

Fastest (CRISTIÁ; MONETTI, 2009) is a flexible, efficient and automatic implementation of the TTF developed conjointly by CIFASIS and Flowgate Consulting. Currently, Fastest automates test suite definition and test case derivation for unit testing. Fastest receives a Z specification in LaTeX format using the CZT package (COMMUNITY Z TOOLS, 2009). Then, the user has to enter a list of the operations to test, as well as the tactics to apply to each of them. In a third step Fastest automatically generates the testing tree of each operation. After the trees are generated, the user can browse them and their test classes, and he/she can prune any node, both manually or automatically. Once the user is done with pruning, he/she can instruct Fastest to find one abstract test case for each leaf in

[1]Test templates can also be called *test suites*, *test classes* or *test objectives*.

all the test trees. Although the method to find abstract test cases has proved to be quite automatic, it is worth to say that it does not guarantee to find abstract test cases for all test objectives. In those cases, the engineer can help Fastest to find a test case by issuing a rather straightforward command. The user can export all the results –testing trees, test classes and abstract test cases– in LaTeX format.

Fastest was envisioned as a client-server application. The main reason for thinking in a distributed system came from the realization that calculating abstract test cases from test objectives in large projects could be a hard computing problem, but highly parallelizable as well. Then, a scalable application using the idle computer power present in a corporate network, became an appealing option. However, in such a large project there is shared information –such as the definition or parametrization of some testing tactics, test cases already calculated, theorems that help to prune testing trees, etc.– that all the clients and servers should be able to access. Hence, a typical Fastest installation has a data server that is known to all other processes, some client processes and a number of testing servers.

4 COMPARISON: CASE STUDIES

This ongoing work is the result of a joint effort between two institutions from Brazil and Argentina. The cooperation started by comparing the MBT techniques used at each institution. The second step was to define a common problem to work with so both groups had a common workbench. The first problem was proposed by INPE's group. CIFASIS researchers then developed a Z model and applied Fastest in order to generate test cases. INPE researchers had already a Statecharts model and the test cases generated by the GTSC environment. The comparison was then extended to two more examples: one proposed by INPE and the other proposed by CIFASIS.

The same methodology was followed in all case studies. First, the party proposing the problem delivers an informal, natural language requirements specification. Second, the other party starts to write a formal model. Third, if the party writing the model finds some problem (incompleteness, inconsistency) or misunderstands the requirements, the other party will send corrections/explanations. Fourth, when the model has been finished the party who wrote it will apply its MBT methodology and tool.

In the next sections, the three case studies are described and some models are presented.

4.1 EXP – OBDH Communication Protocol

The first problem proposed by INPE was the software that will be embedded into an astrophysical experiment, hereafter called EXP, computer of a Brazilian scientific satellite. A proprietary protocol was specified for the communication between EXP and the On-Board Data Handling (OBDH) computer. OBDH is the satellite platform computer to process platform and payload information and to generate and format data that has to be transmitted to Ground Stations.

OBDH sends one out of nine commands at a time to EXP, which returns an answer to OBDH. Each command must arrive within certain time constraints. Commands ask EXP to perform some operations or to return some data about its state. There are simple commands and there are more complex commands to transmit scientific data acquired by the payload, to dump EXP's computer memory, to load data sent by OBDH, etc.

Figures 4.1, 4.2 and Table 4.1 show a Statecharts model, the flat FSM[1] produced by GTSC and test cases derived according to two test criteria, respectively. The model in Figure 4.1 refers to a piece of the entire software embedded into EXP computer. In Figure 4.2, each state of the flat FSM is actually a configuration of two active BASIC states of the Statecharts model at a certain instant of time. Besides, it shows only a piece of the flat FSM (the entire model has 16 states and 61 transitions). Also note that some transitions do not have an explicit output (in these cases, outputs are null).

It is important to emphasize that the Statecharts model in Figure 4.1 was derived based on a perspective of the design of the software product. When test cases are derived from design documents, some authors define the technique as *gray box* testing (ABDURAZIK; OFFUTT, 2000).

[1] In all state-transition diagrams representing FSMs in this work, a state depicted by two circles, one inner surrounded by other outer, is the initial state of the FSM.

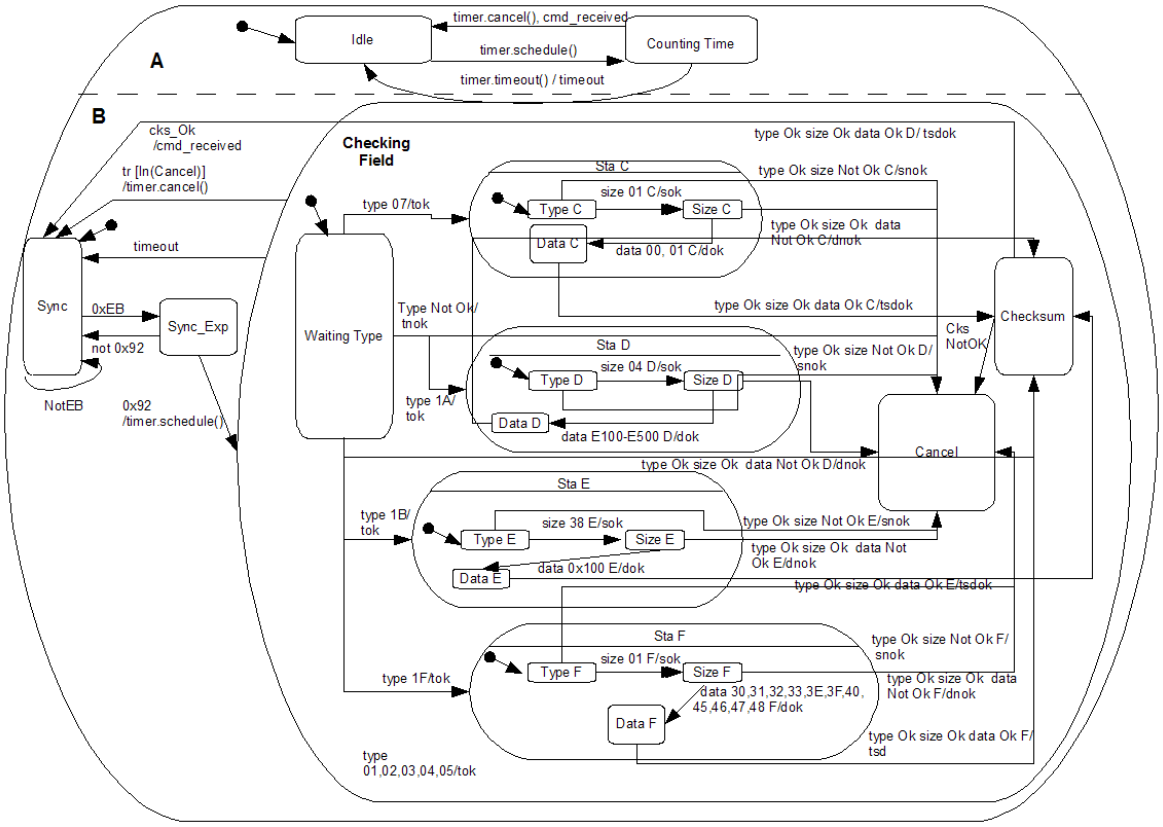

Figure 4.1 - Statecharts model of the command recognition component of APEX software.

Table 4.1 - Samples of test cases for the model in Figure 4.1.

Criteria	Test Cases
all-transitions	{(0xEB/null, 0x92/null, Type01/tok, Cks_Ok/null), (0xEB/null, 0x92/null, Type07/tok, Size01_C/sok, Data00_C/dok, TypeOkSizeOkDataOK_C/tsdok), (0xEB/null, 0x92/null, Type1B/tok, Size38_E/sok, Data0x100_E/dok, TimerTimeout/null), (0xEB/null, 0x92/null, Type1F/tok, TimerTimeout/null), ...}
all-simple-paths	{(0xEB/null, 0x92/null, Type02/tok, Cks_Ok/null), (0xEB/null, 0x92/null, Type07/tok, Size01_C/sok, Data01_C/dok, TypeOkSizeOkDataOK_C/tsdok, Cks_NotOk/null,), (0xEB/null, 0x92/null, Type1B/tok, Size38_E/sok, Data0x100_E/dok, TypeOkSizeOkDataOK_E/tsdok, Cks_Ok/null), (0xEB/null, 0x92/null, Type1F/tok, Size01_F/sok, Data33_F/dok, TypeOkSizeOkDataOK_F/tsdok, TimerTimeout/null), ...}

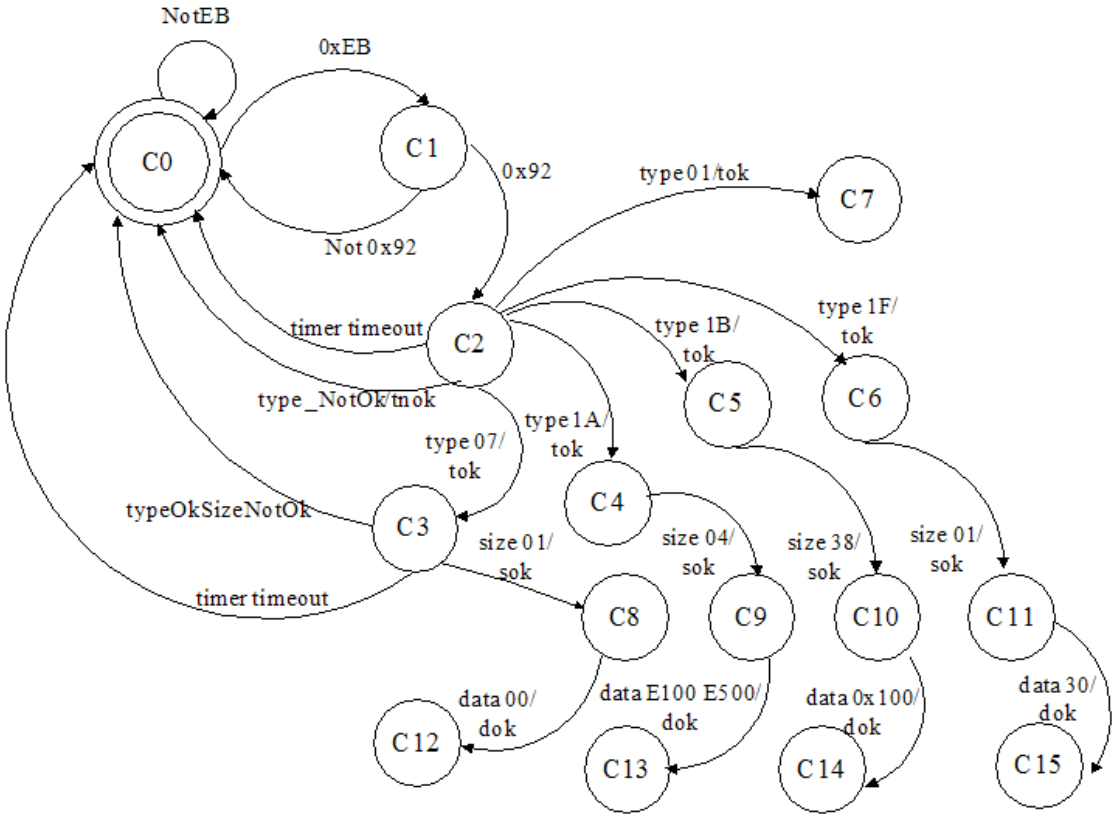

Figure 4.2 - A piece of the flat FSM obtained by GTSC for the model in Figure 4.1.

4.2 A Simple Scheduler

This case study was proposed by CIFASIS borrowing it from (UTTING; LEGEARD, 2007). The problem is about the basic operations of a simple scheduler. The environment can buffer processes for latter execution and can withdraw a process from the waiting list. On the other hand, the scheduler can swap between the active process and one other process ready for execution. The idea behind this problem was twofold: (a) to propose a simpler problem, and (b) to apply the techniques to a somewhat less reactive system.

Figure 4.3 shows the Statecharts model for the Scheduler case study where *new1* *new2*, ... refer to the same operation: new(pid)[2]. This had to be done because the GTSC environment does not accept non-determinism. The same remark applies for

[2]pid = process identifier

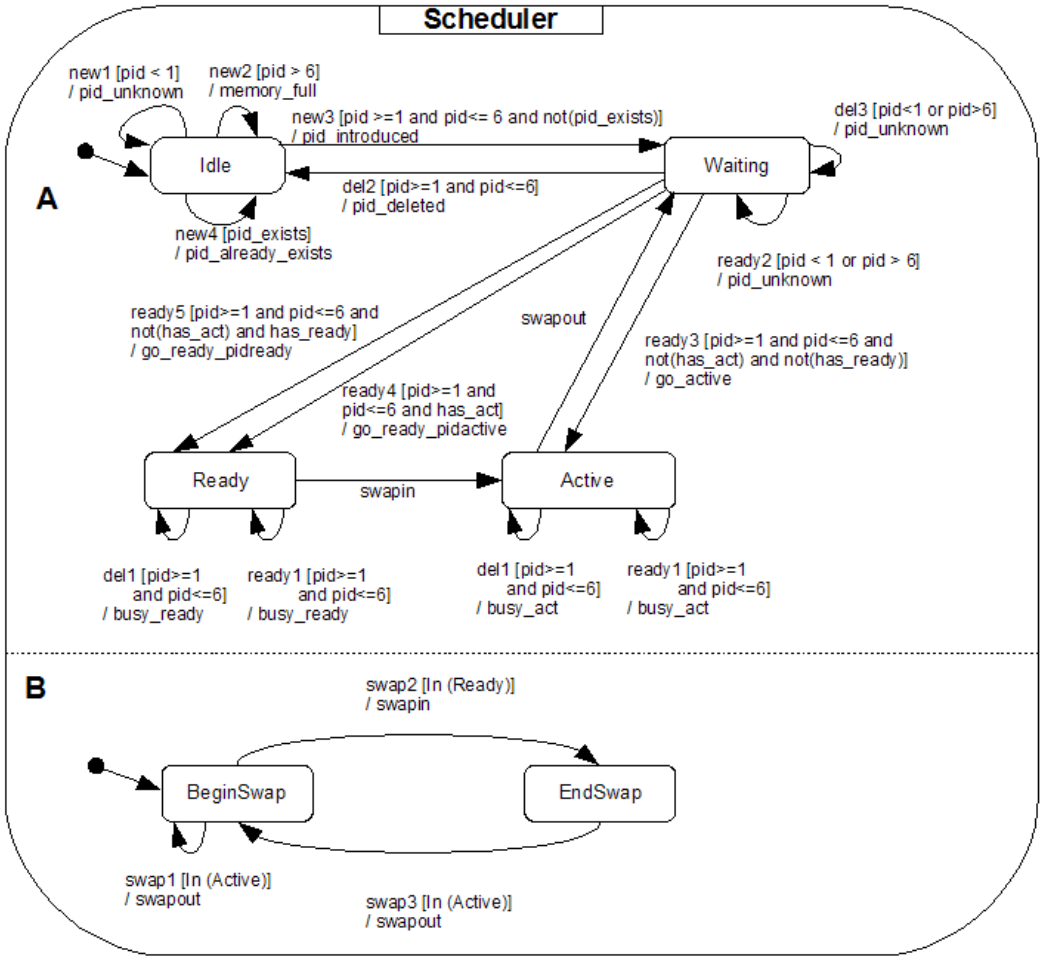

Figure 4.3 - Statecharts model for the Scheduler case study.

the other events in the model. Another observation is that there are also guarding conditions within the labels of the transitions. Four variables exist as shown in Table 4.2.

In order to generate test cases, the values of the variables shown in Table 4.2 shall be chosen. For this purpose, the traditional black box testing technique Equivalence Partitioning (MATHUR, 2008) was used and five classes were identified according to Table 4.3. For each one of these classes, GTSC produces a different flat FSM and, based on such model, test cases are generated. Figures 4.4, 4.5, 4.6, 4.7, 4.8 show the Flat FSMs for the classes 4, 9, 12, 16, and 20 respectively. Recall that each state of each flat FSM is actually a configuration of two active BASIC states of the

Table 4.2 - Variables for the Scheduler case study.

Variable	Type	Meaning
pid	integer	Process identification
pid_exists	boolean	A process has already been created in the system with this pid
has_act	boolean	There is an active process
has_ready	boolean	There are ready processes

Table 4.3 - Equivalence classes for the Scheduler case study.

Class	pid	pid_exists	has_act	has_ready	Remark
4	$pid < 1$	false	true	true	$pid = 0$
9	$1 \leq pid \leq 6$	false	false	false	$pid = 1$
12	$1 \leq pid \leq 6$	false	true	true	$pid = 5$
16	$1 \leq pid \leq 6$	true	true	true	$pid = 6$
20	$pid > 6$	false	true	true	$pid = 7$

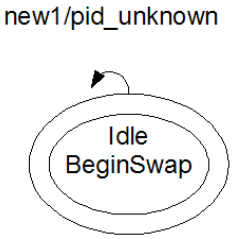

Figure 4.4 - Flat FSM for class 4.

Statecharts model.

4.3 SWPDC

The third case study was a software product specified and developed in the scope of the *Qualidade do Software Embarcado em Aplicações Espaciais* (QSEE - Quality of Space Application Embedded Software) research project at INPE (SANTIAGO et al.,

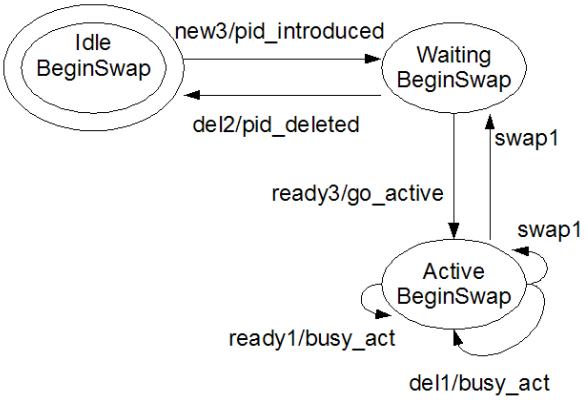

Figure 4.5 - Flat FSM for class 9.

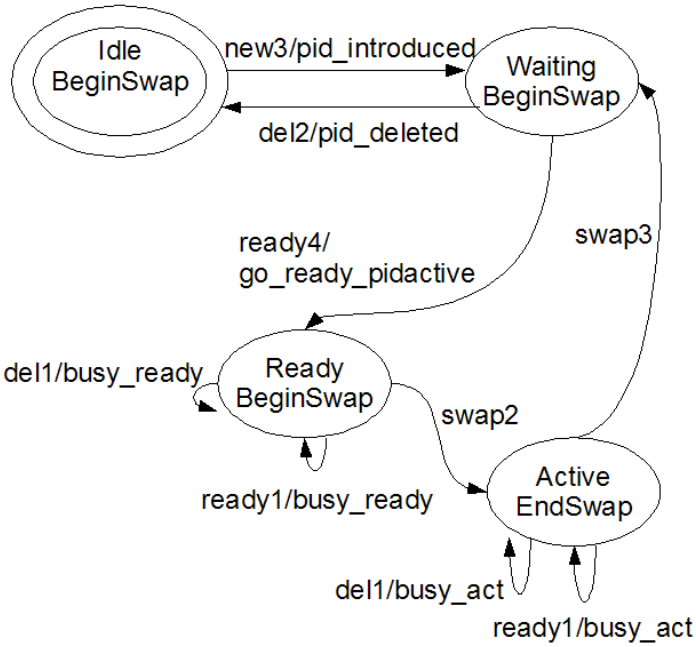

Figure 4.6 - Flat FSM for class 12.

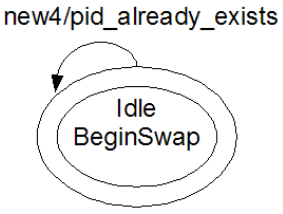

Figure 4.7 - Flat FSM for class 16.

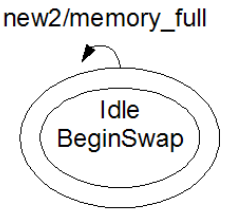

Figure 4.8 - Flat FSM for class 20.

2007). SWPDC is the software embedded into the Payload Data Handling Computer (PDC) and it is the hardest problem of all analyzed in this work. Although similar in conception to the case study described in Section 4.1, SWPDC is much more complex because it handles not only scientific and dump data (e.g. housekeeping data), but also accomplishes data memory management, implements flow control mechanisms, etc. Data transmission is more complex since SWPDC has to keep record of the last transmitted frame because OBDH can ask it again if some problem during transmission was detected.

Differently from the model for the EXP case study (Section 4.1) in which the design perspective was the option, in the SWPDC case study, a usage scenario approach was taken into account in order to generate the models. Hence, 20 usage scenarios were chosen and consequently 20 sets of Statecharts models were elaborated. This is a typical black box testing (MATHUR, 2008) by means of MBT.

The 20 usage scenarios cover several situations such as transmission of scientific, dump and housekeeping data, initiation of the system, changing of software parameters, distinction if the system is being powered on or if a reset has occurred and so on. All 20 usage scenarios and their respective models are shown from Figure 4.9 to Figure 4.50. In most cases, due to the hierarchy feature of Statecharts, a single usage scenario has more than one model. Besides, parallelism is addressed in all Statecharts models.

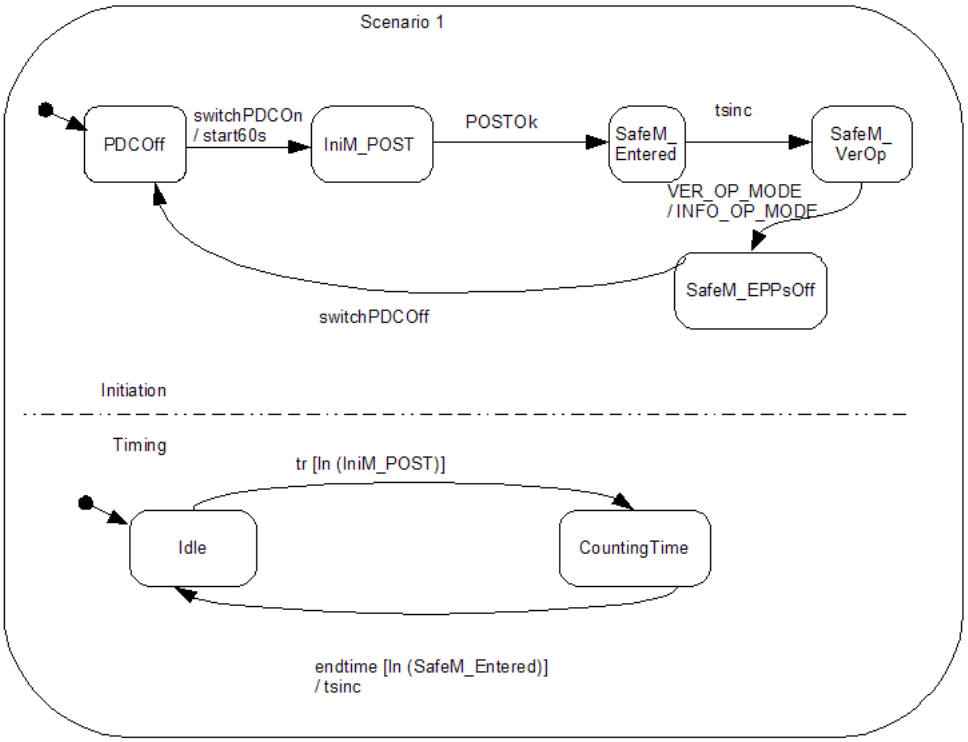

Figure 4.9 - Statechart Model: Scenario 1.

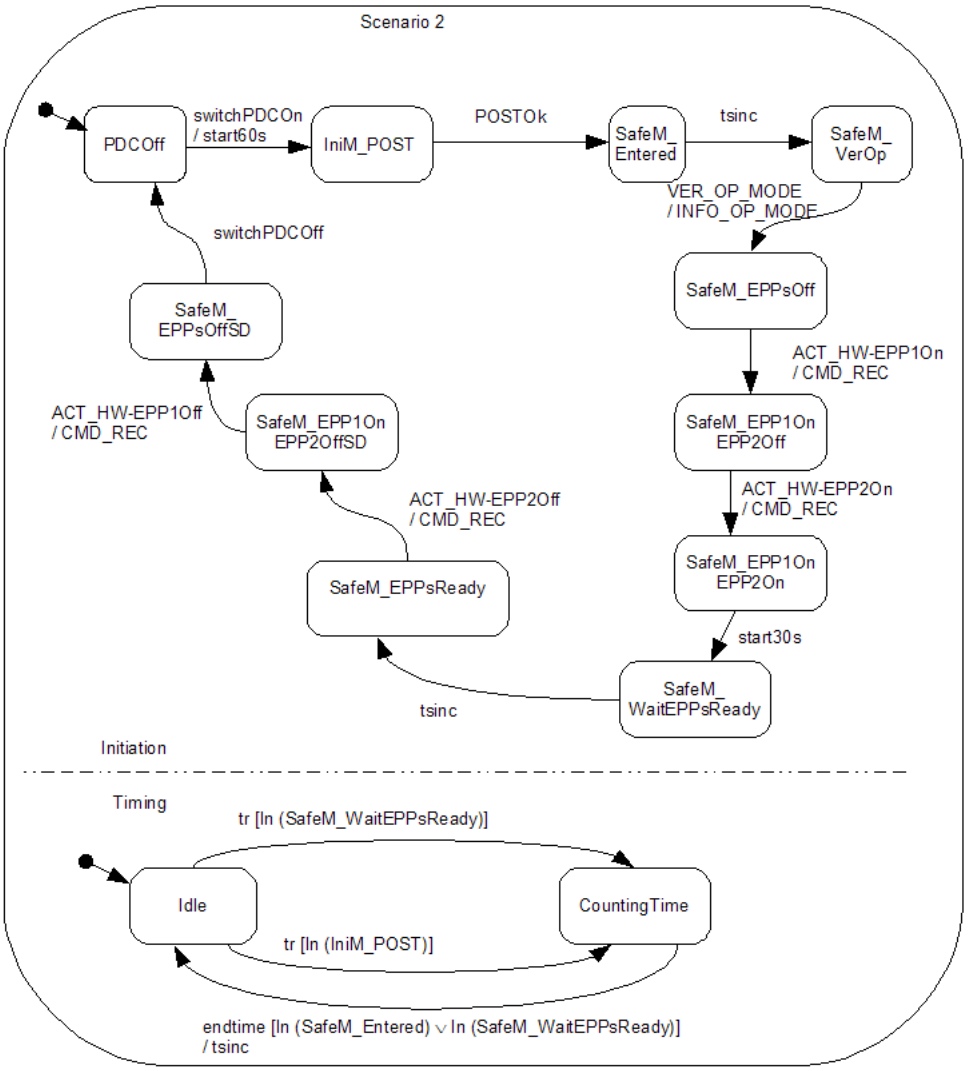

Figure 4.10 - Statechart Model: Scenario 2.

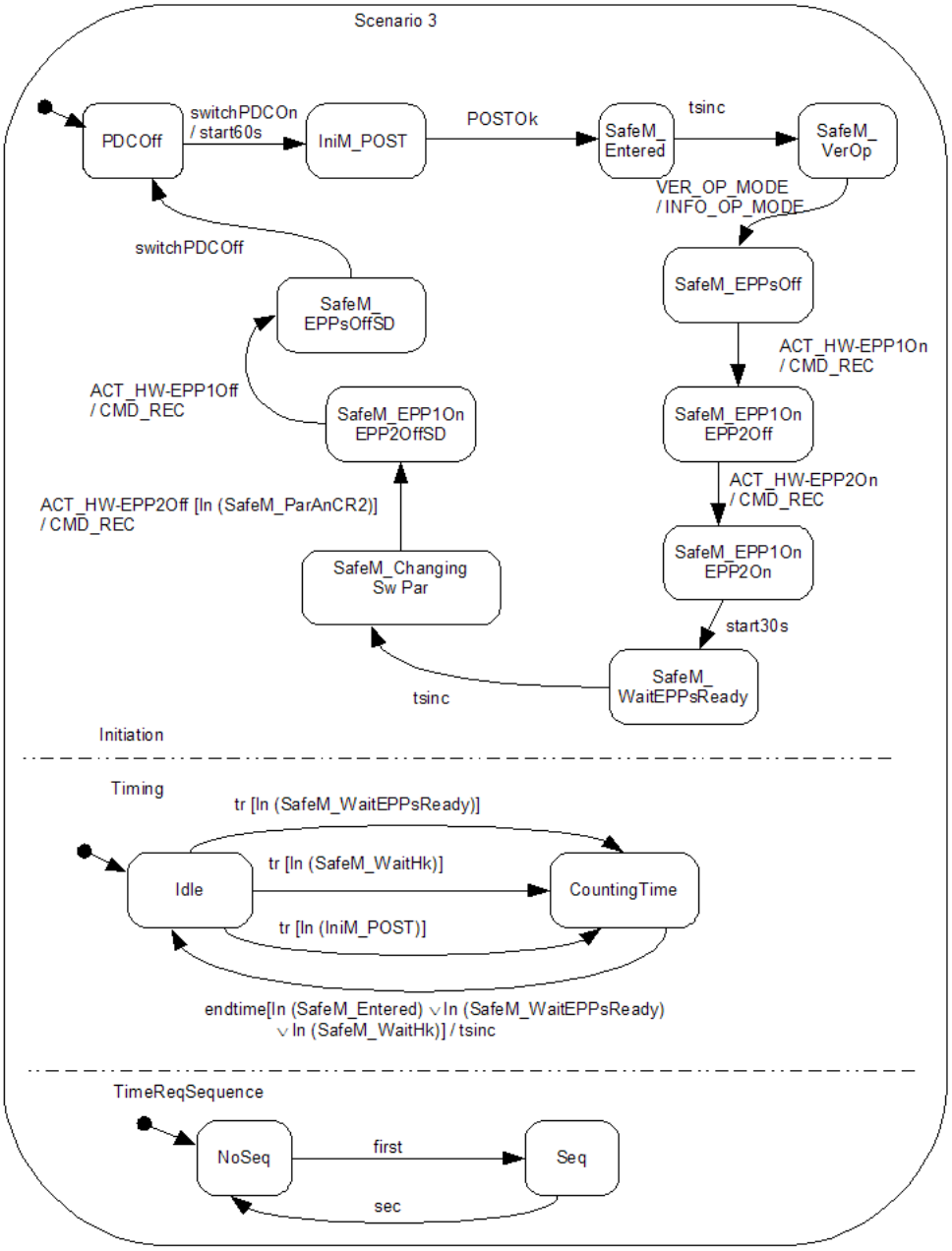

Figure 4.11 - Statechart Model: Scenario 3 (main model).

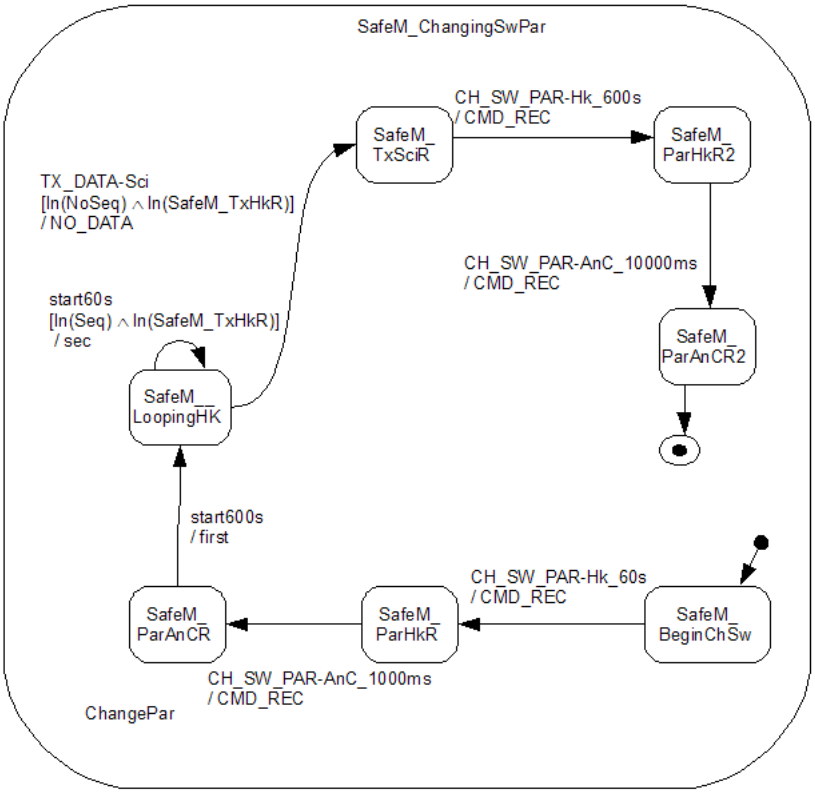

Figure 4.12 - Statechart Model: Scenario 3 (second hierarchy level).

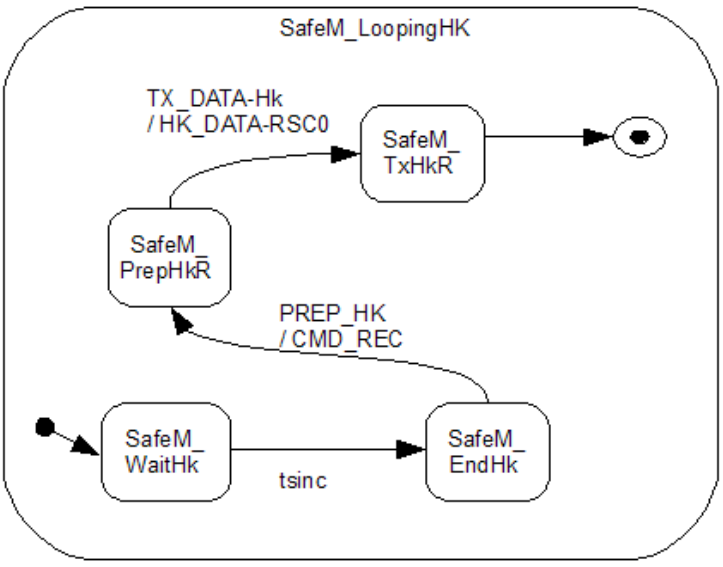

Figure 4.13 - Statechart Model: Scenario 3 (third hierarchy level).

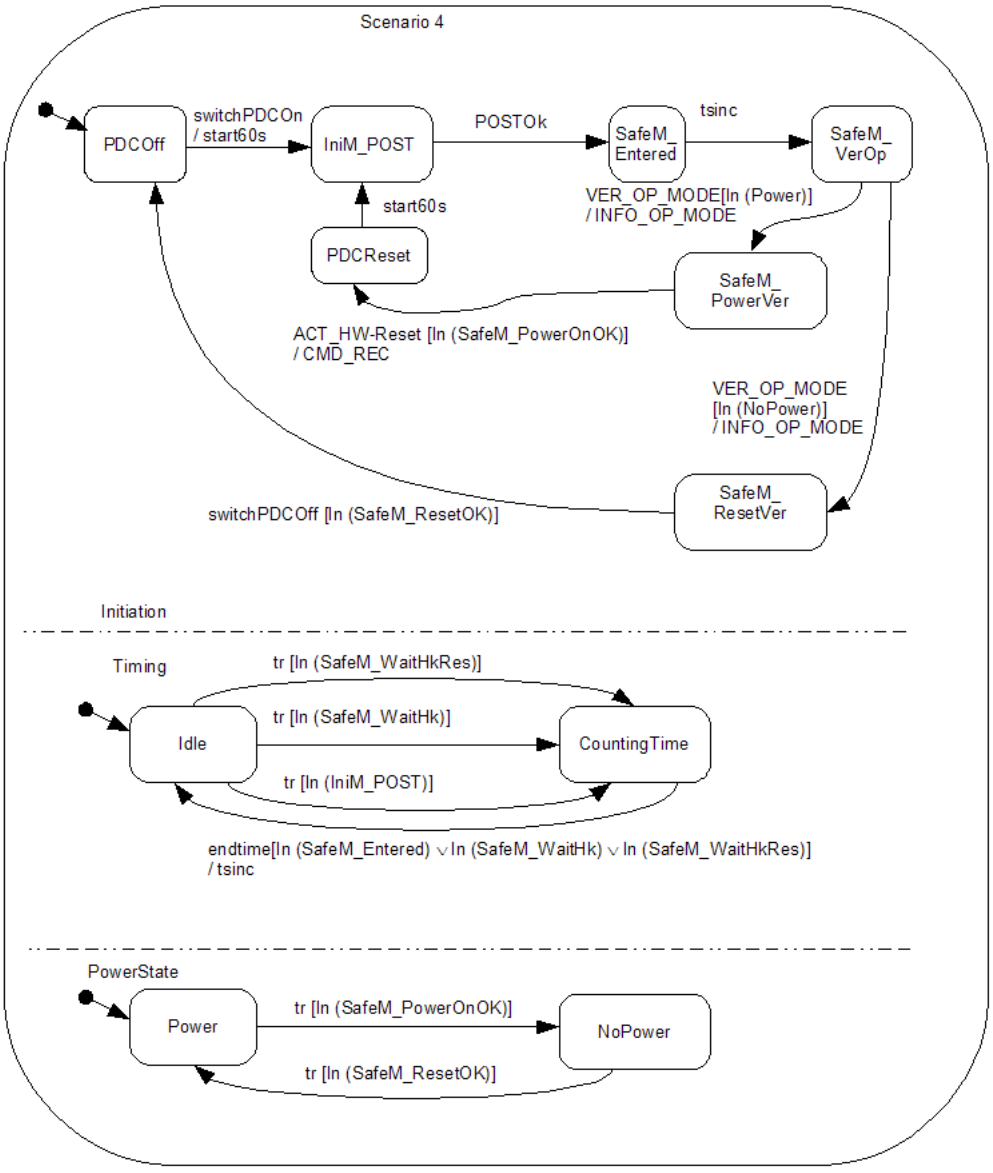

Figure 4.14 - Statechart Model: Scenario 4 (main model).

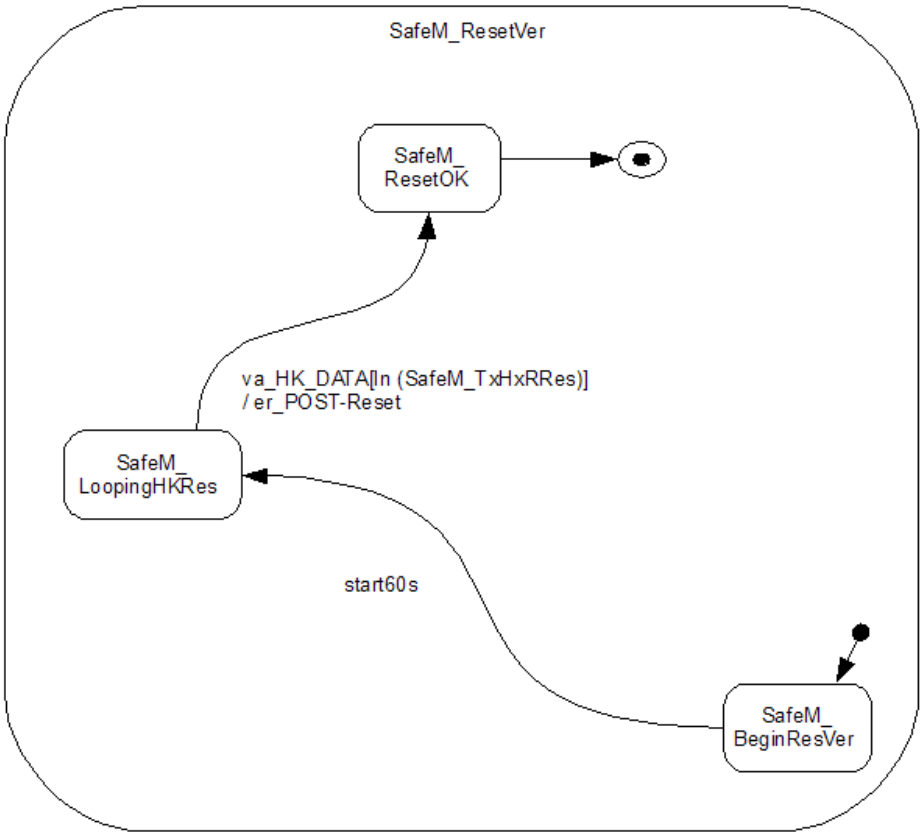

Figure 4.15 - Statechart Model: Scenario 4 (second hierarchy level).

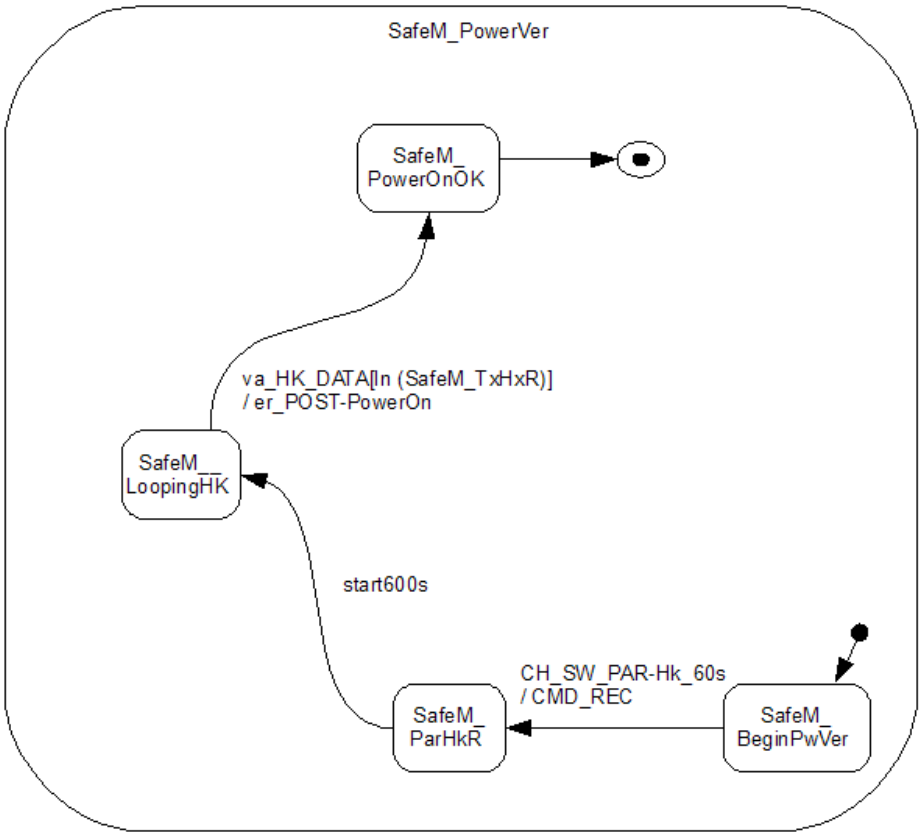

Figure 4.16 - Statechart Model: Scenario 4 (second hierarchy level).

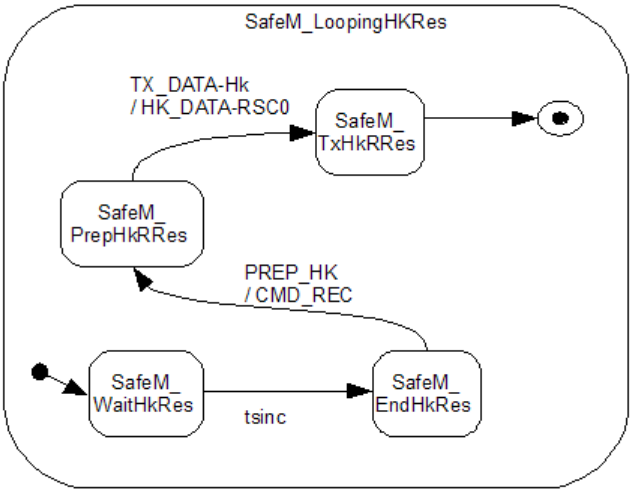

Figure 4.17 - Statechart Model: Scenario 4 (third hierarchy level).

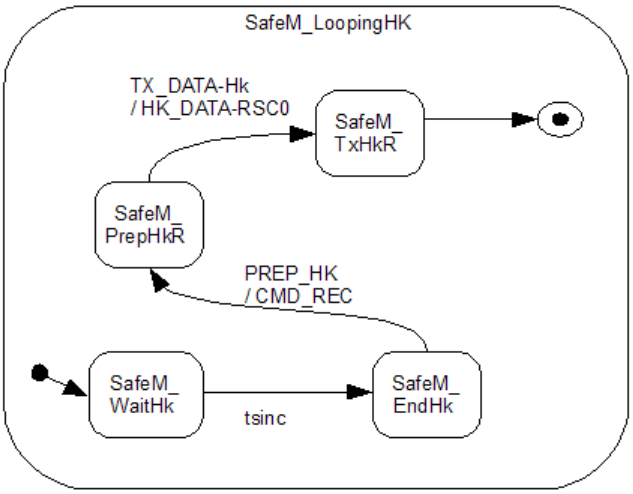

Figure 4.18 - Statechart Model: Scenario 4 (third hierarchy level).

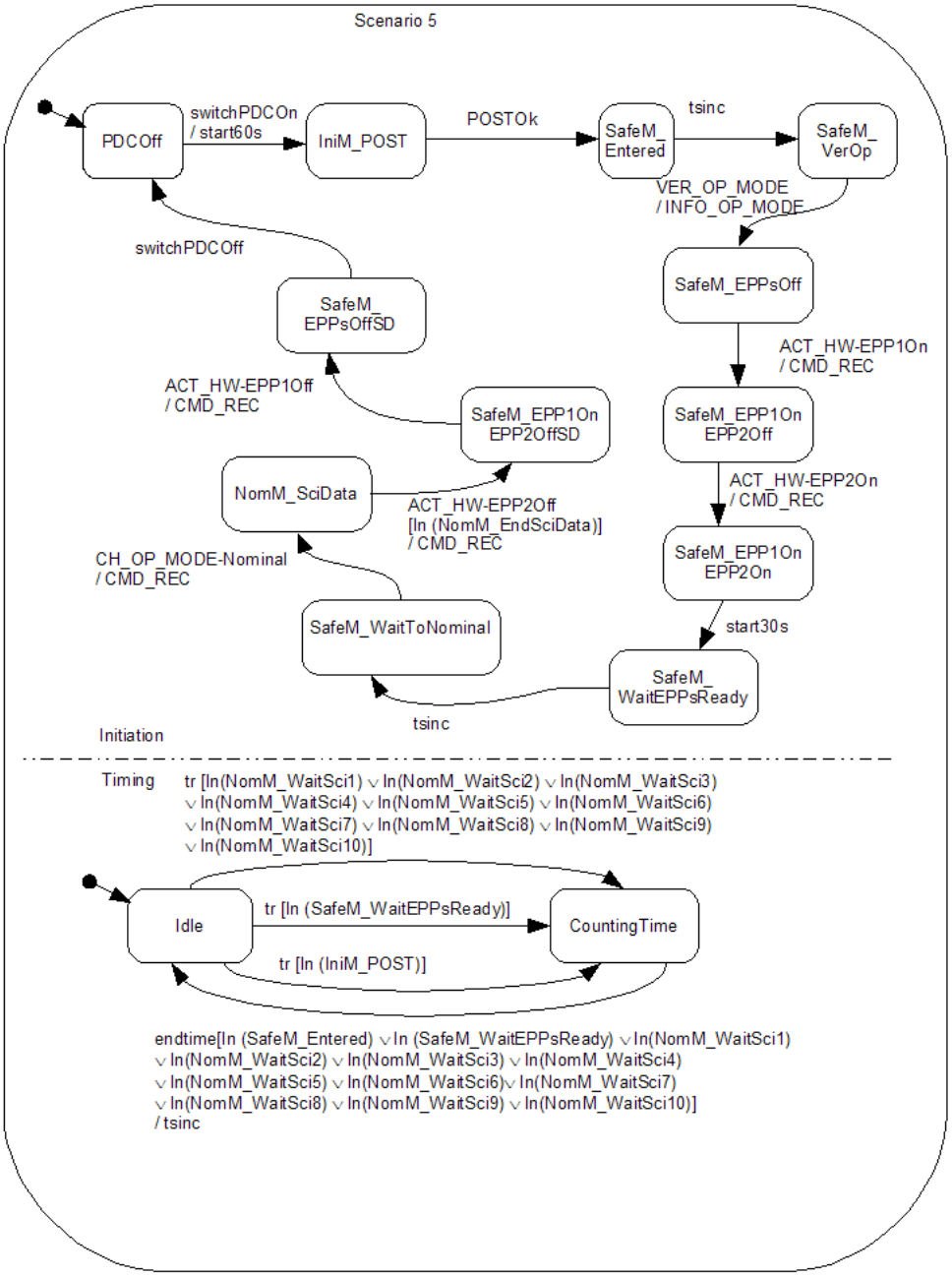

Figure 4.19 - Statechart Model: Scenario 5 (main model).

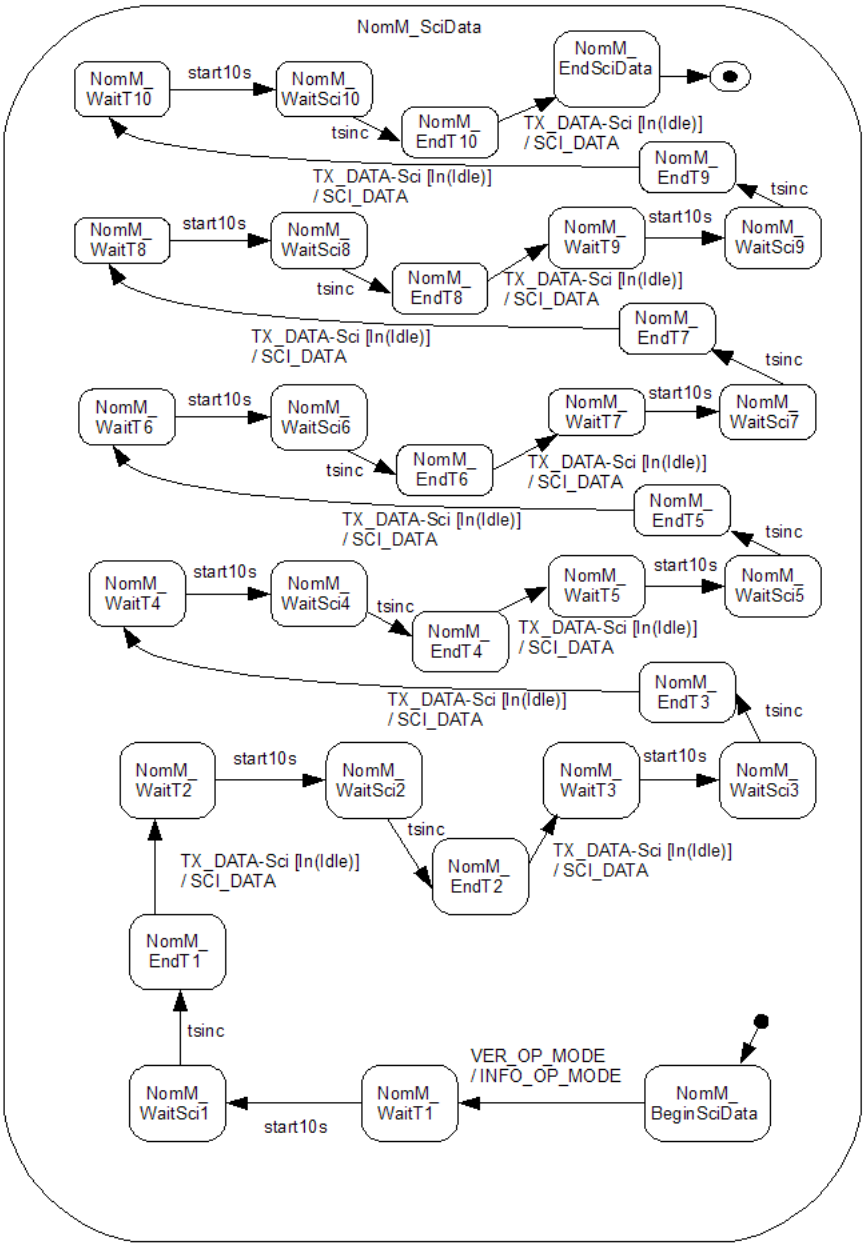

Figure 4.20 - Statechart Model: Scenario 5 (second hierarchy level).

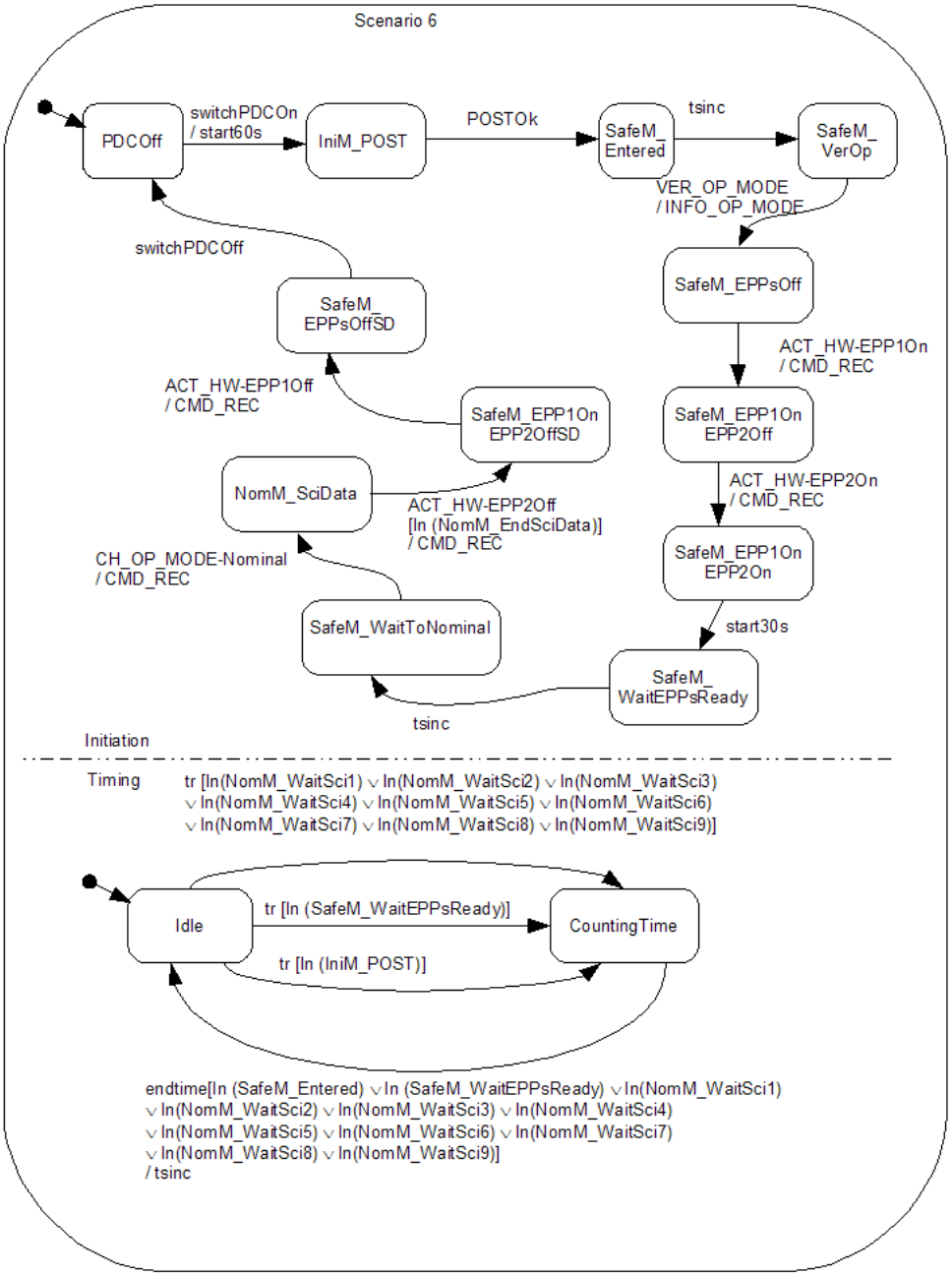

Figure 4.21 - Statechart Model: Scenario 6 (main model).

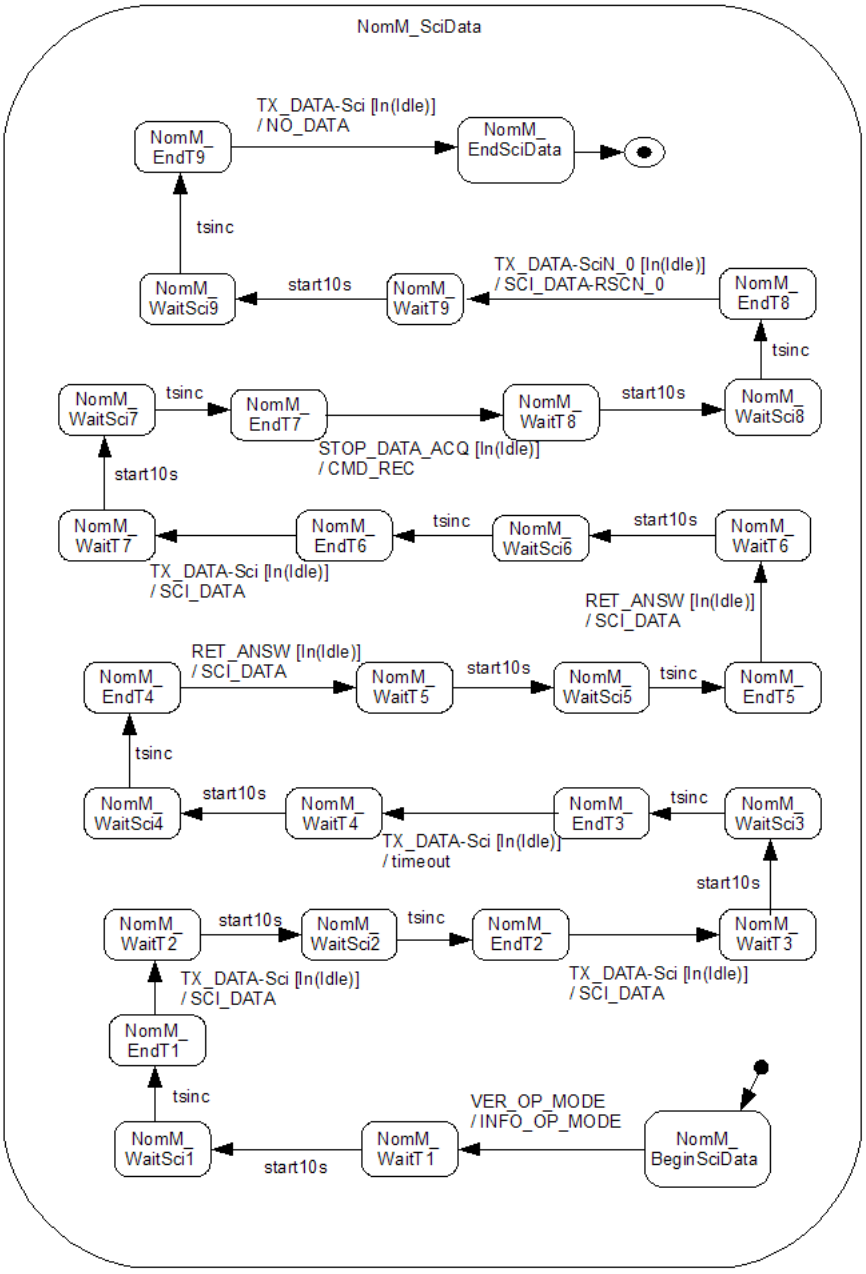

Figure 4.22 - Statechart Model: Scenario 6 (second hierarchy level).

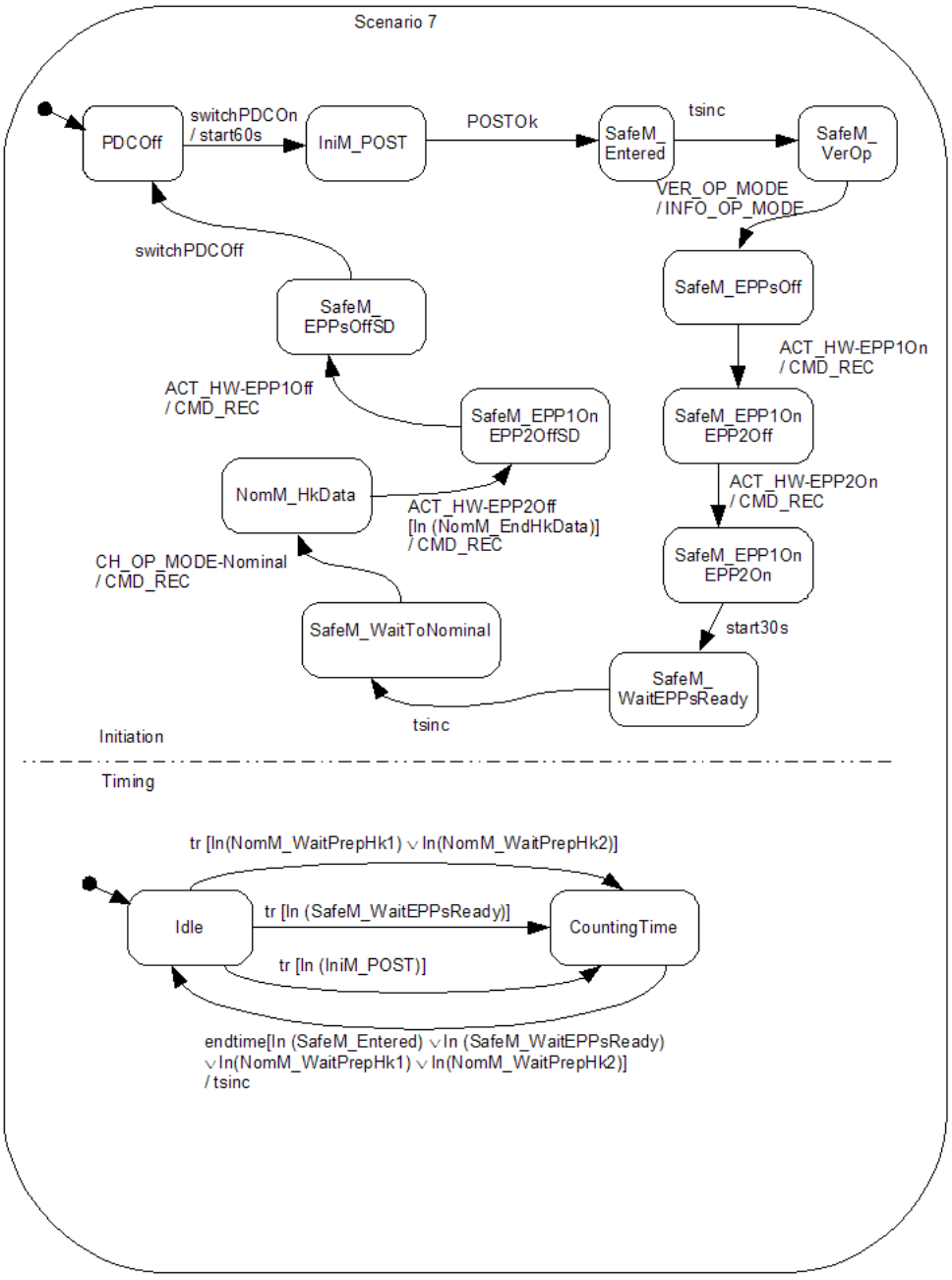

Figure 4.23 - Statechart Model: Scenario 7 (main model).

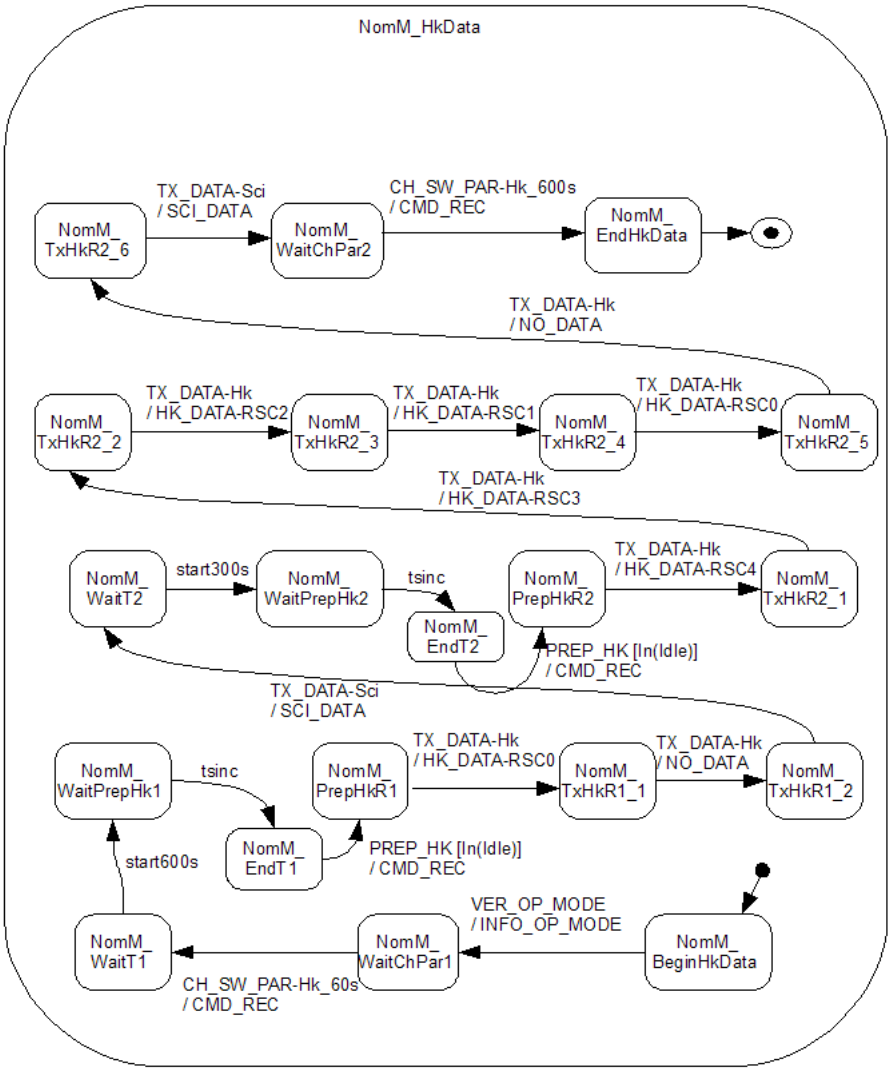

Figure 4.24 - Statechart Model: Scenario 7 (second hierarchy level).

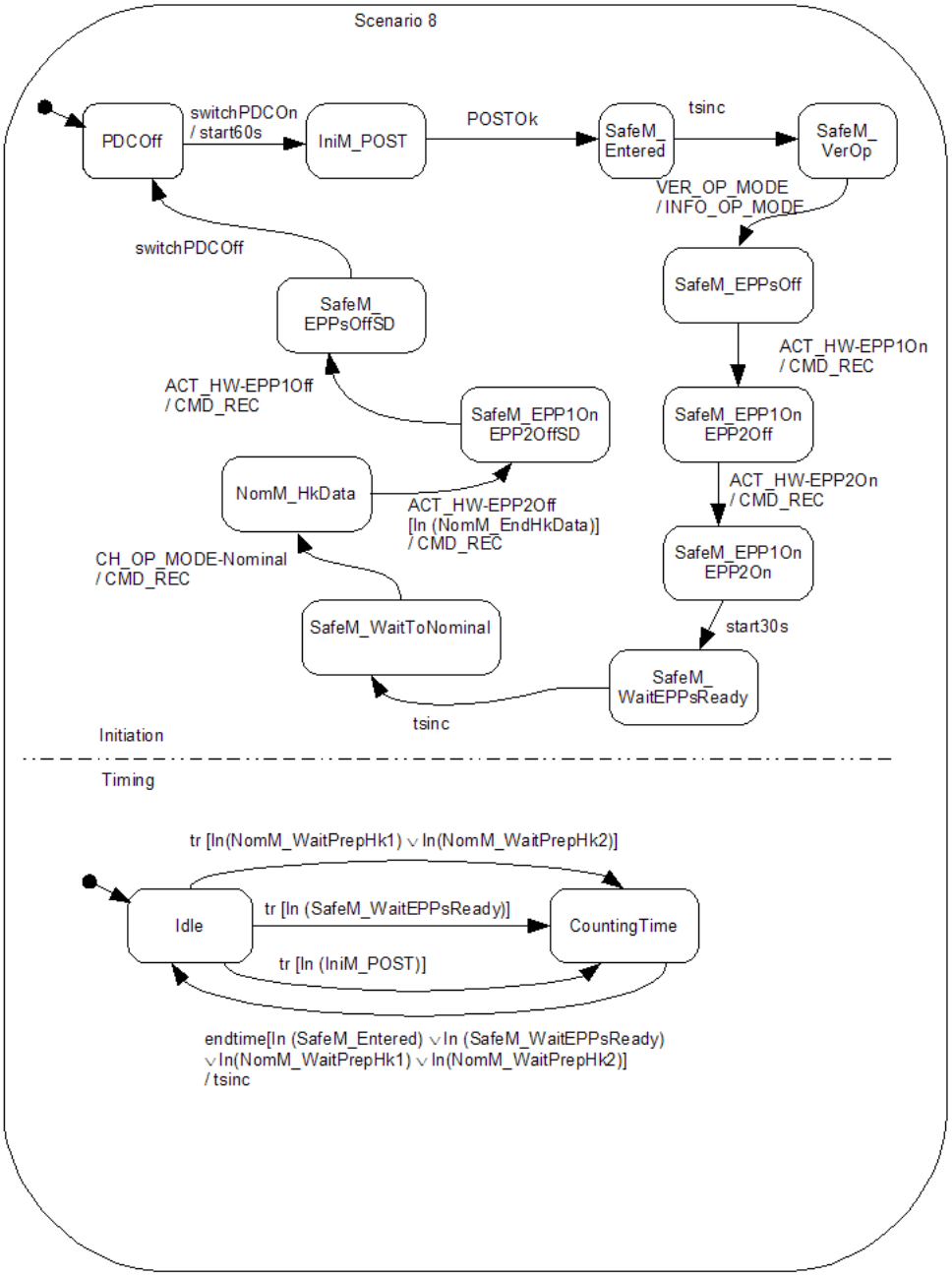

Figure 4.25 - Statechart Model: Scenario 8 (main model).

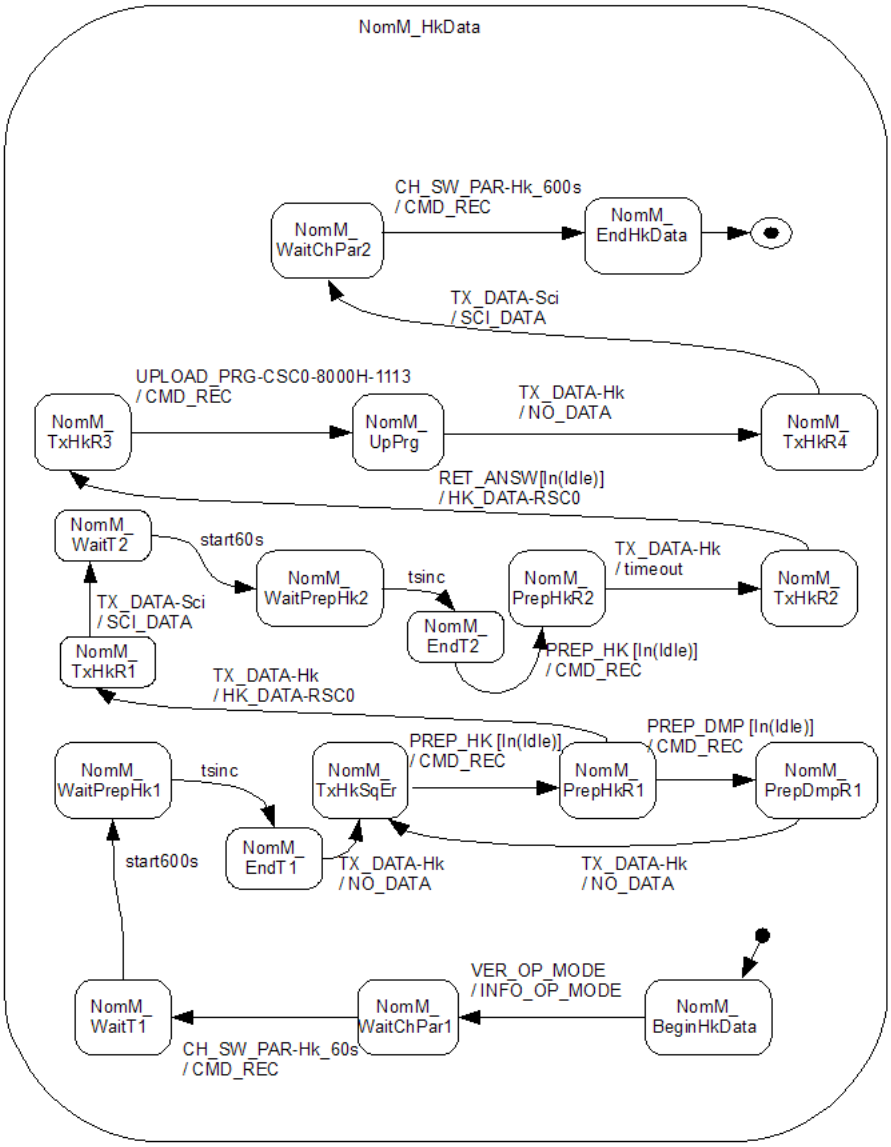

Figure 4.26 - Statechart Model: Scenario 8 (second hierarchy level).

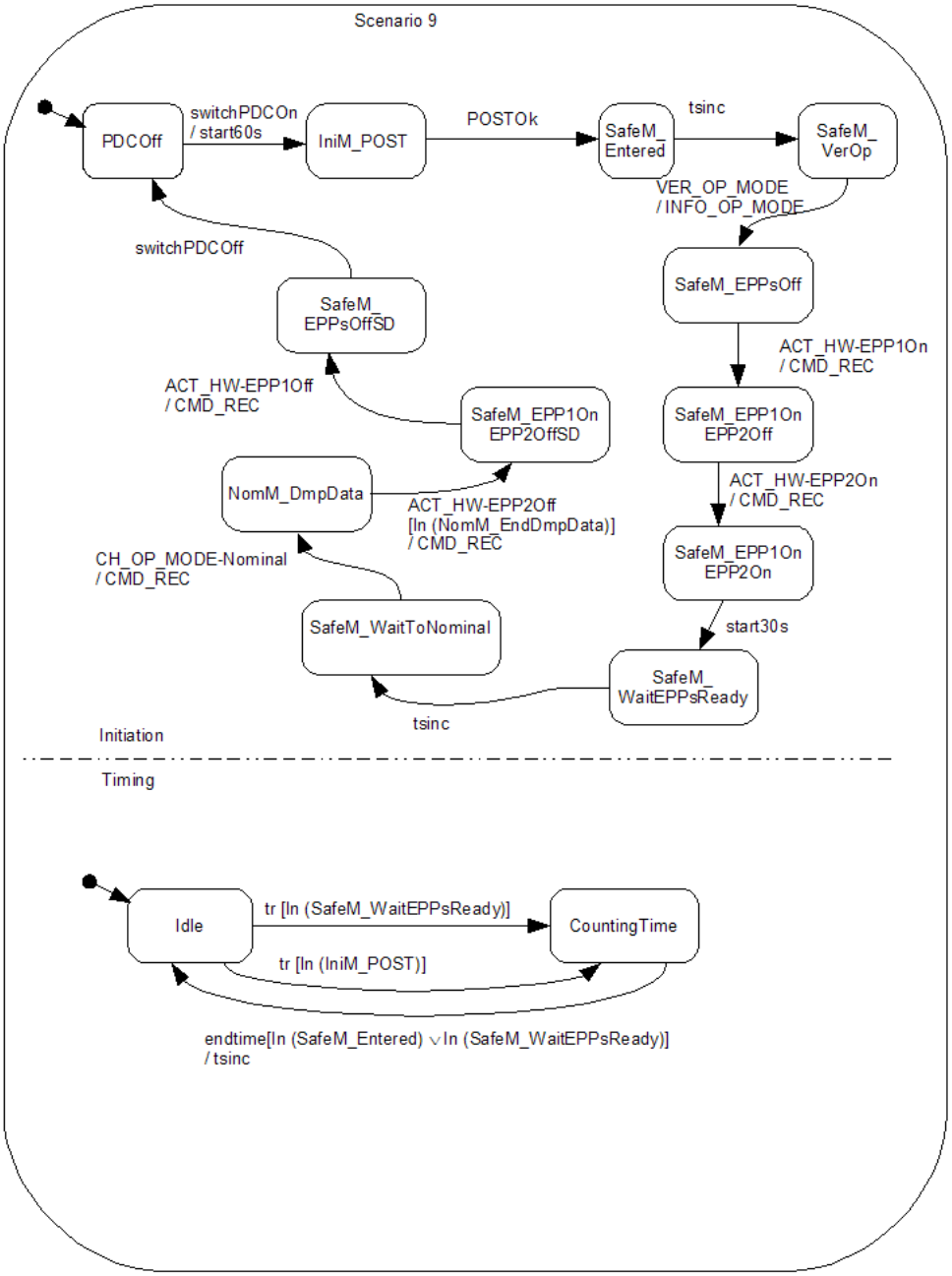

Figure 4.27 - Statechart Model: Scenario 9 (main model).

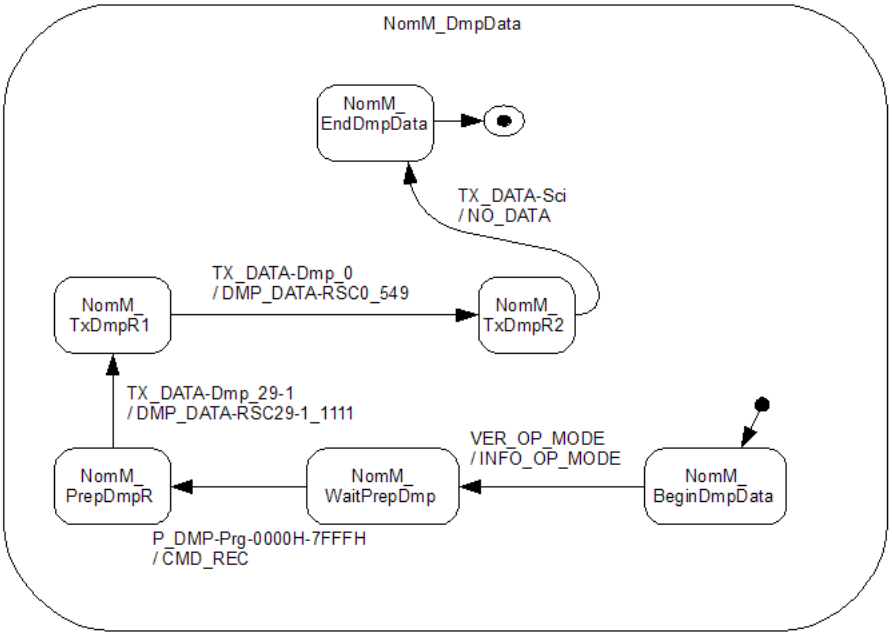

Figure 4.28 - Statechart Model: Scenario 9 (second hierarchy level).

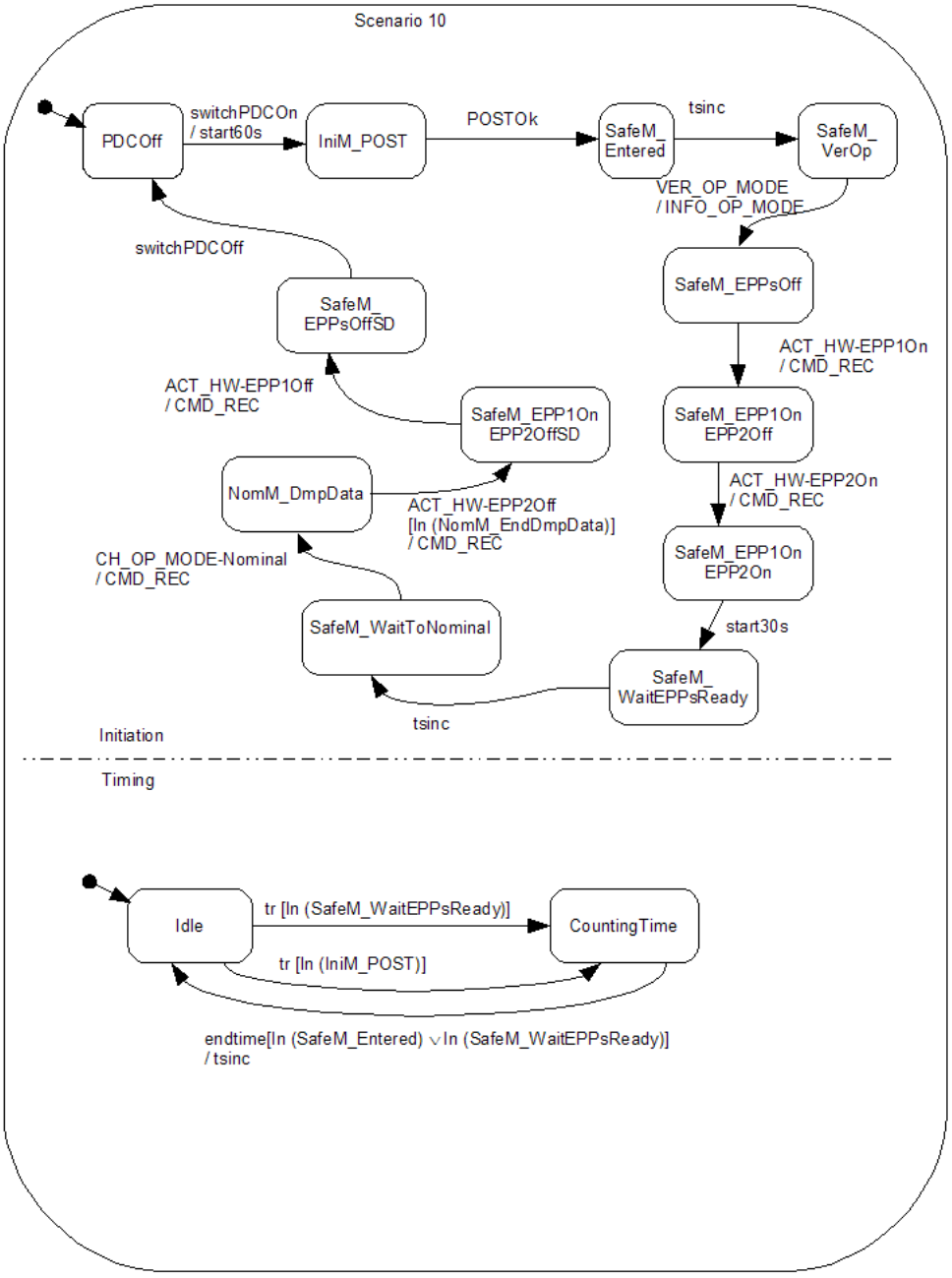

Figure 4.29 - Statechart Model: Scenario 10 (main model).

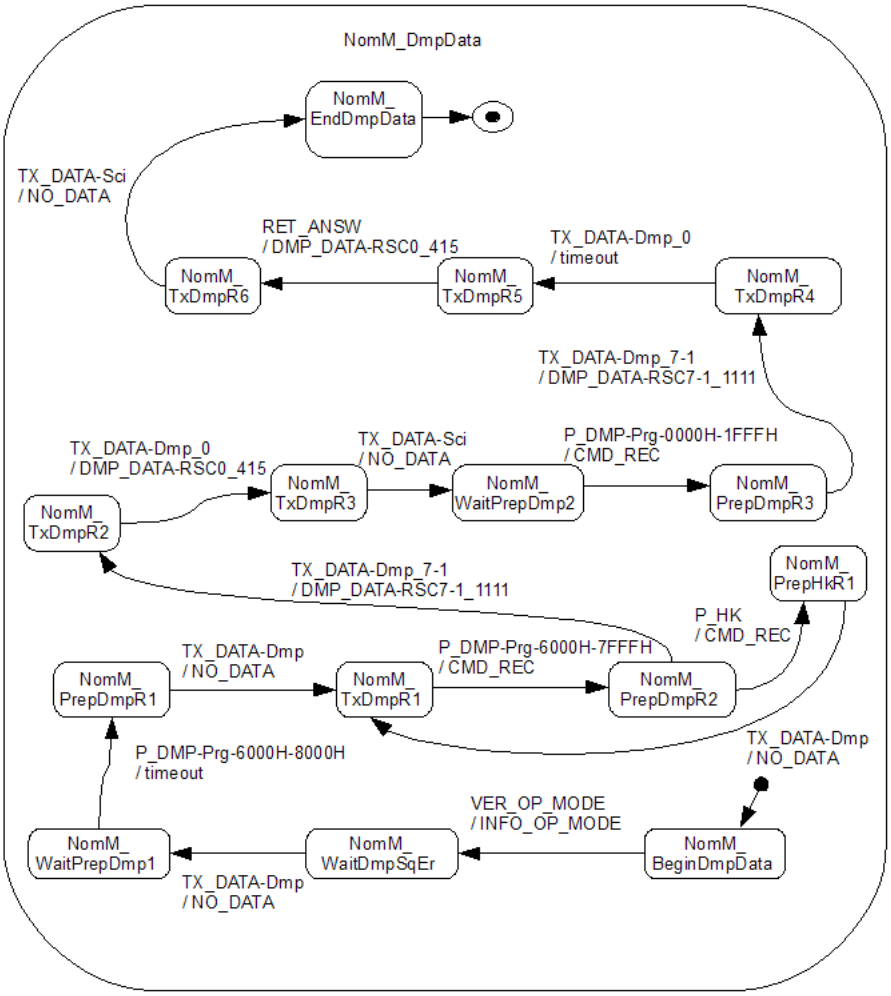

Figure 4.30 - Statechart Model: Scenario 10 (second hierarchy level).

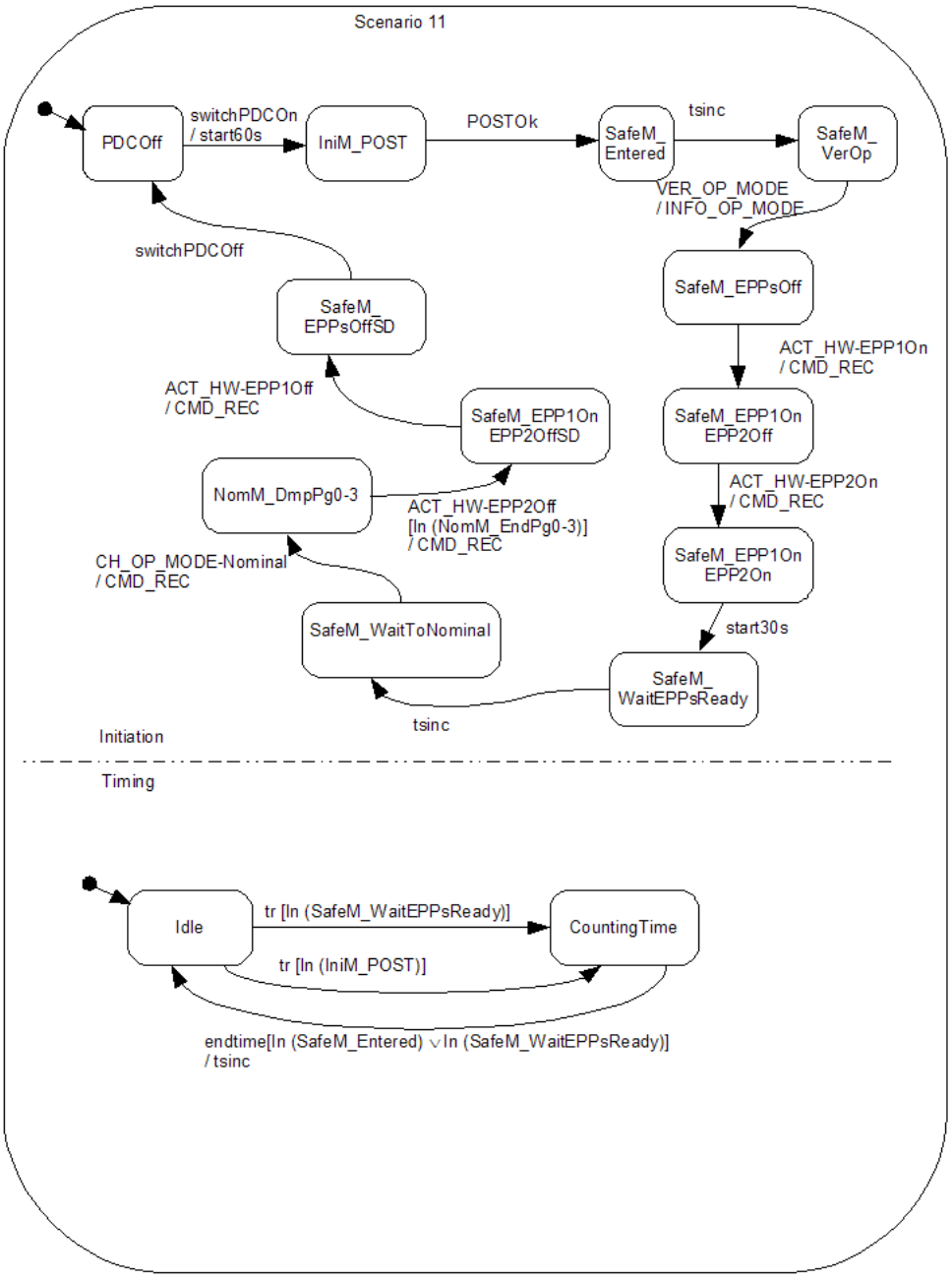

Figure 4.31 - Statechart Model: Scenario 11 (main model).

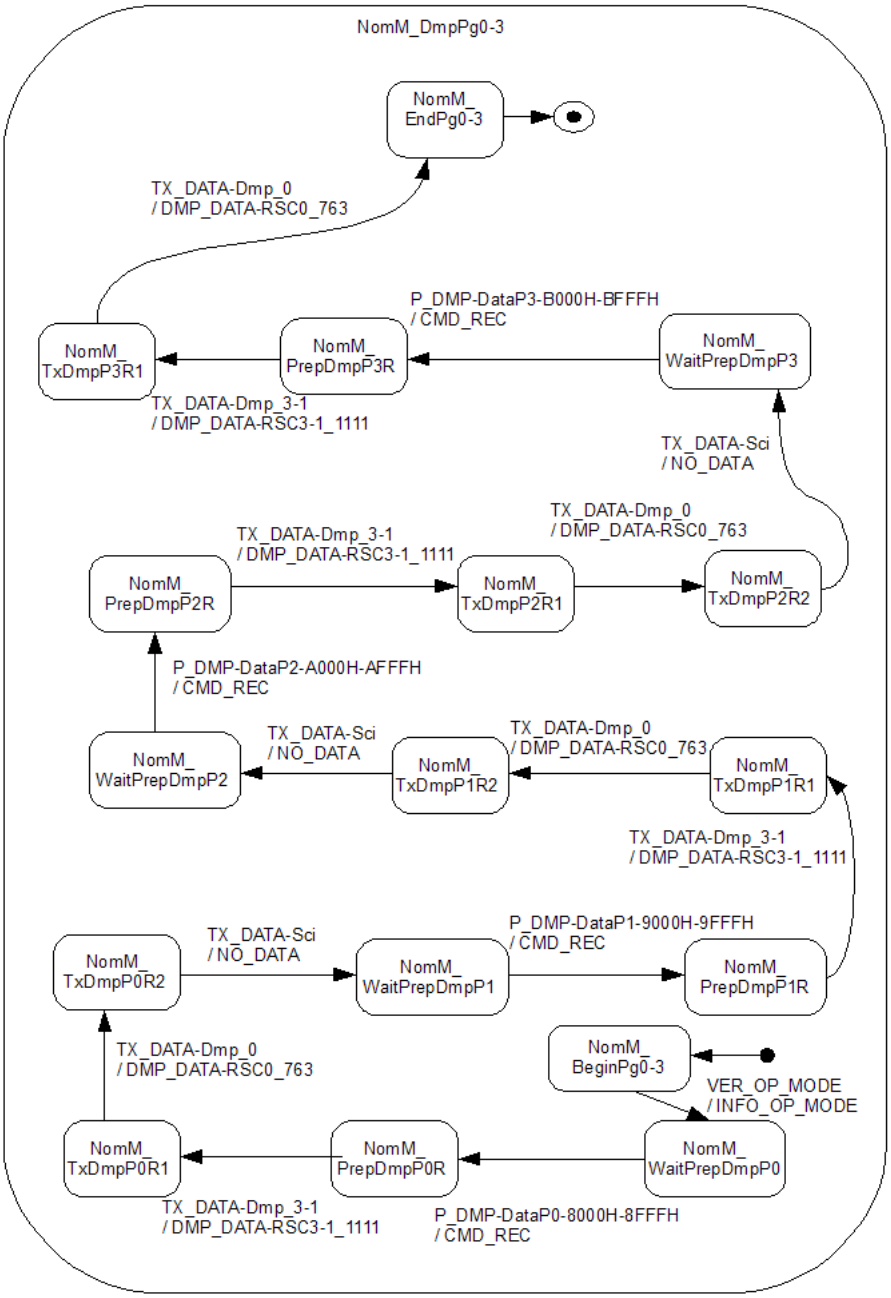

Figure 4.32 - Statechart Model: Scenario 11 (second hierarchy level).

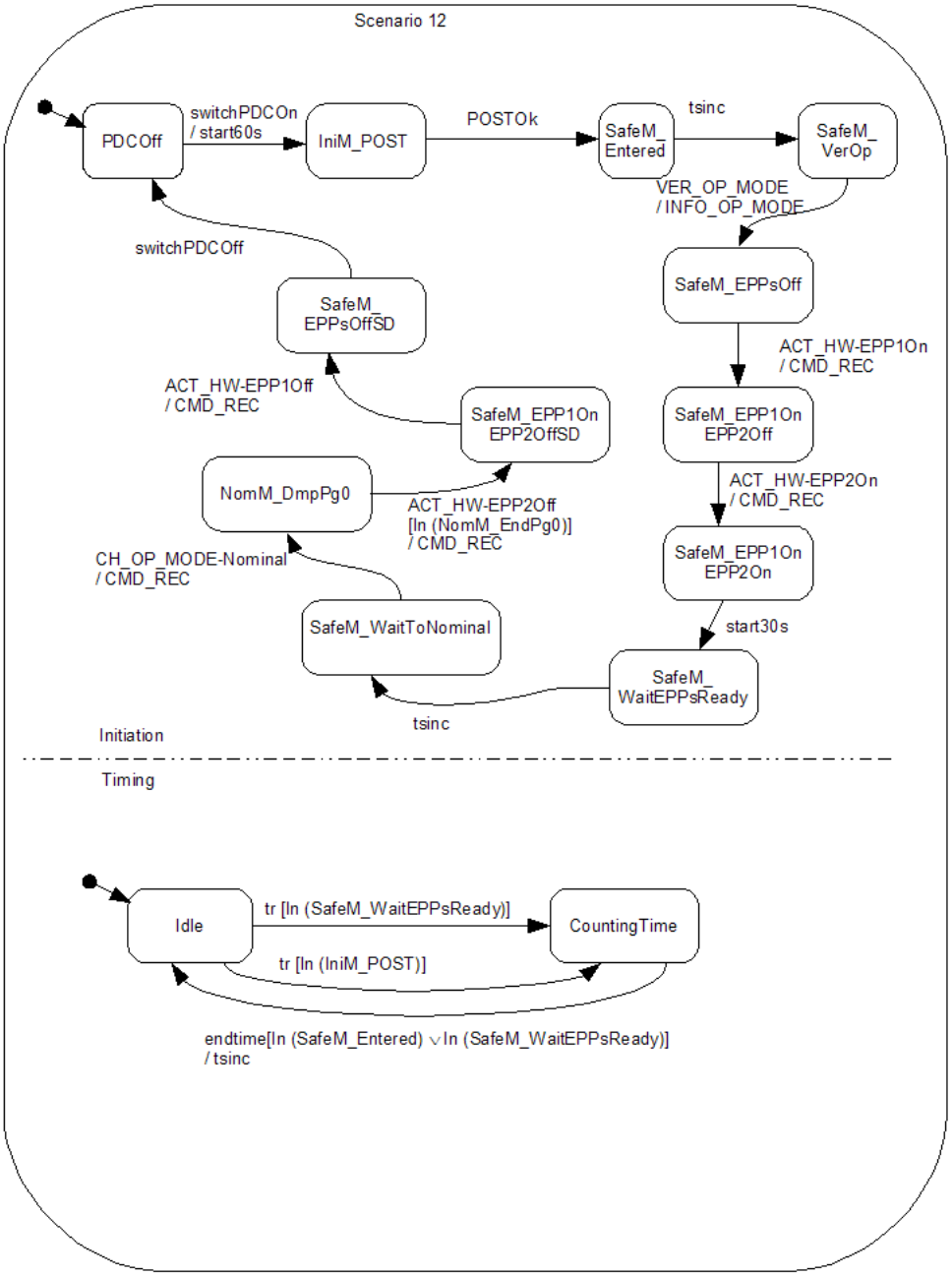

Figure 4.33 - Statechart Model: Scenario 12 (main model).

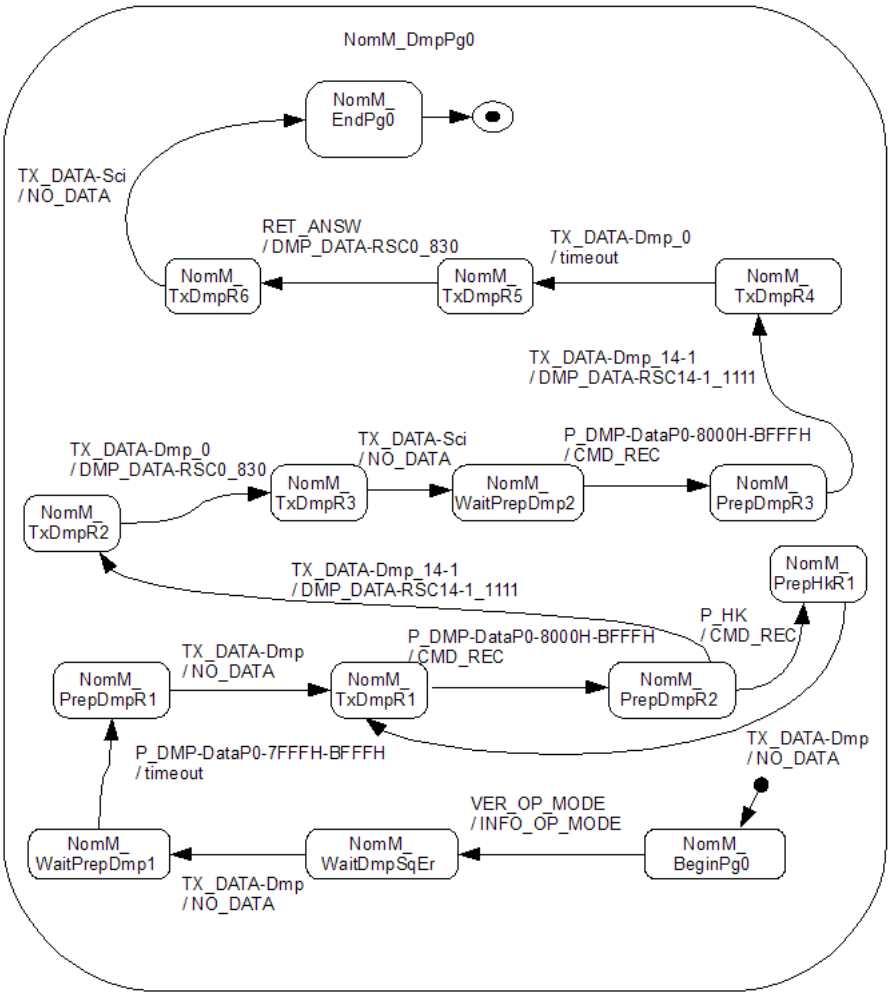

Figure 4.34 - Statechart Model: Scenario 12 (second hierarchy level).

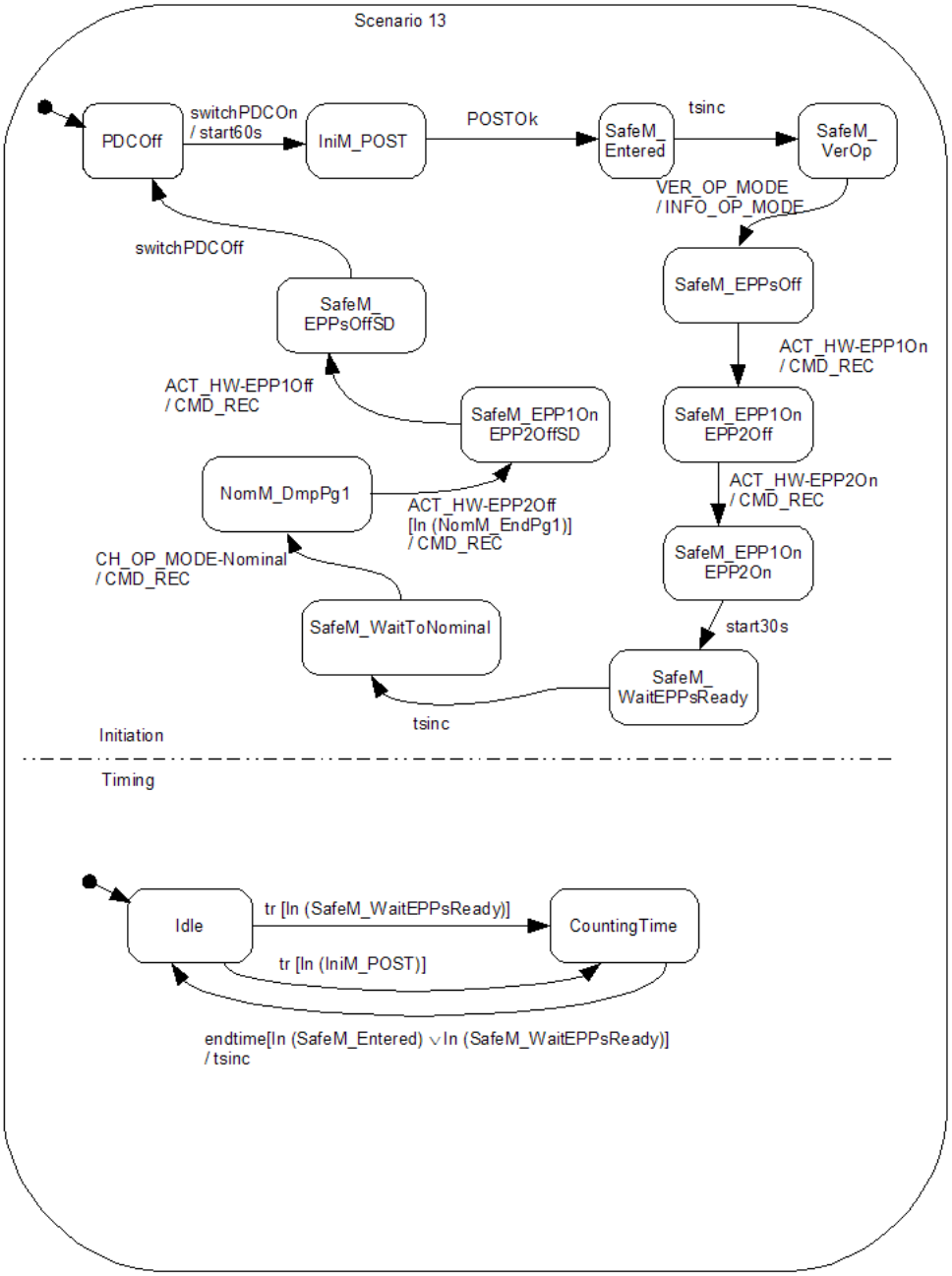

Figure 4.35 - Statechart Model: Scenario 13 (main model).

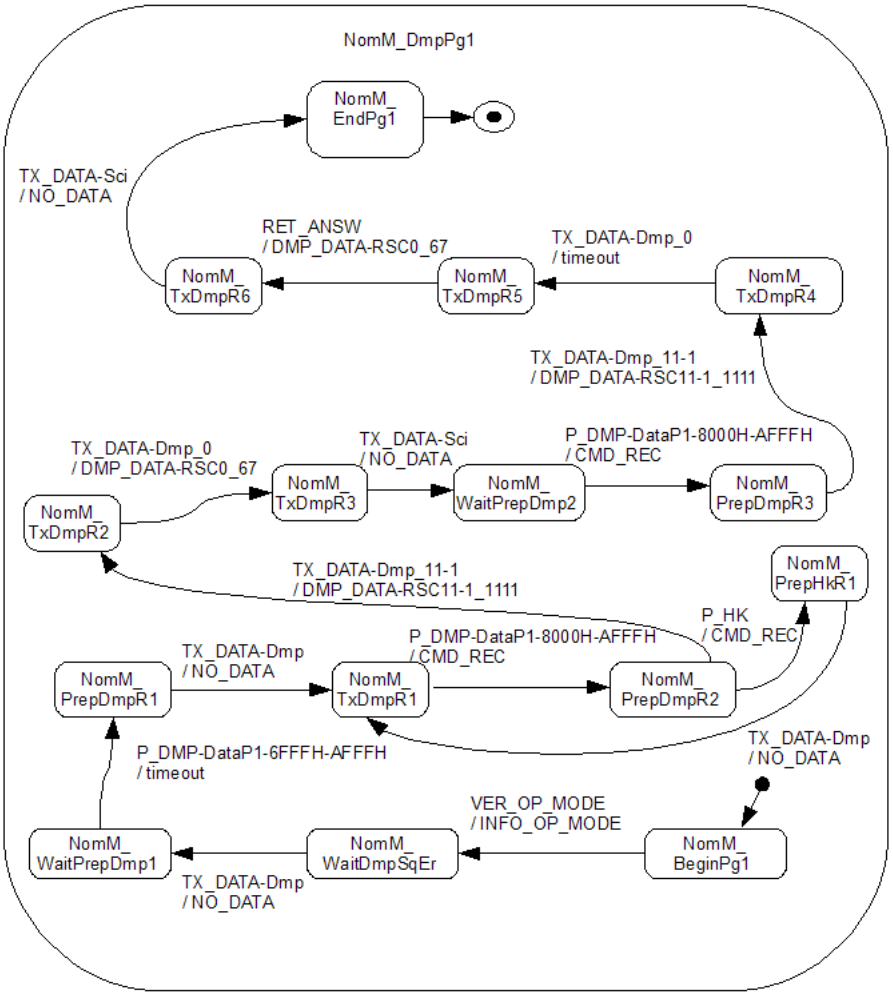

Figure 4.36 - Statechart Model: Scenario 13 (second hierarchy level).

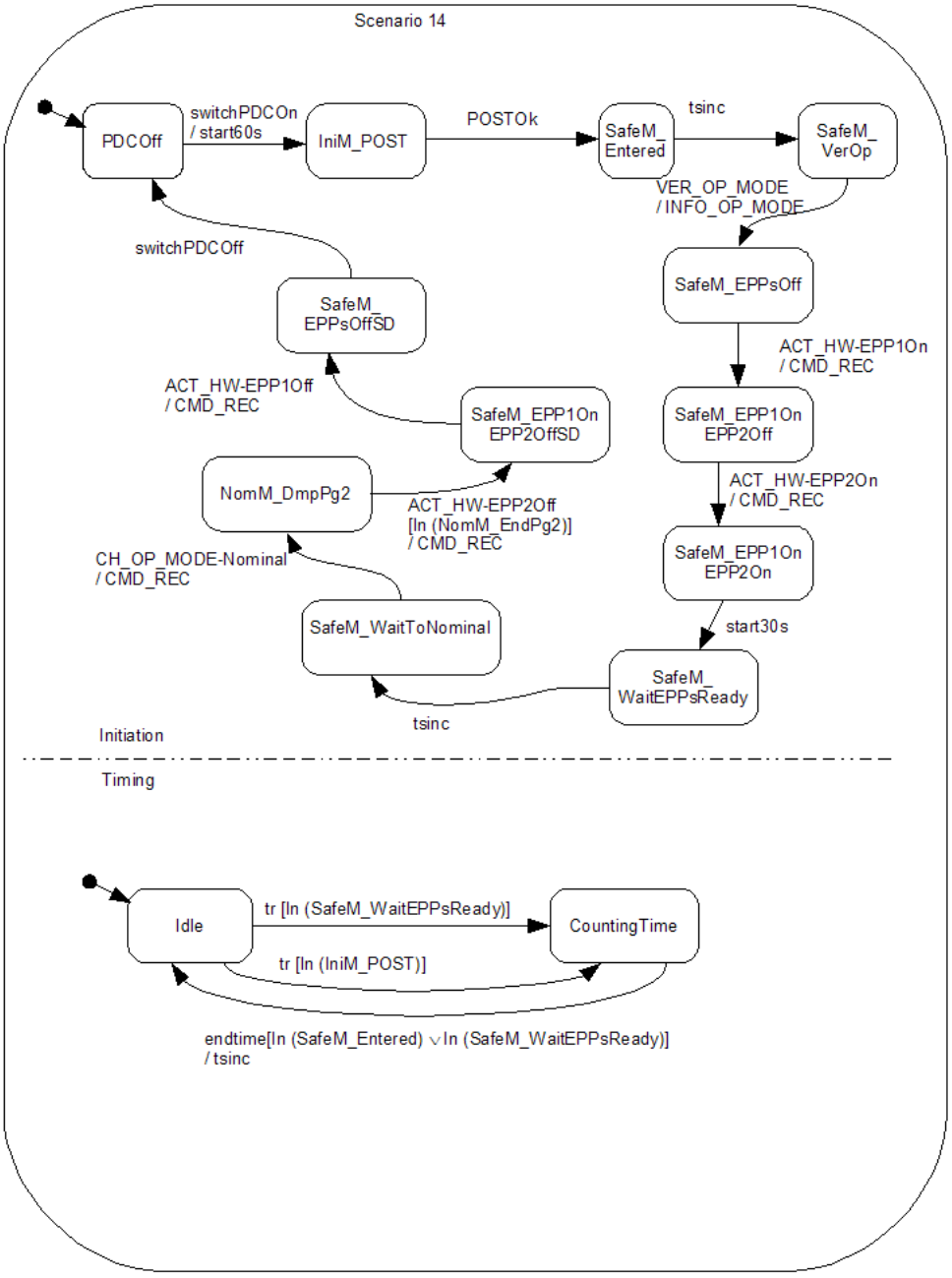

Figure 4.37 - Statechart Model: Scenario 14 (main model).

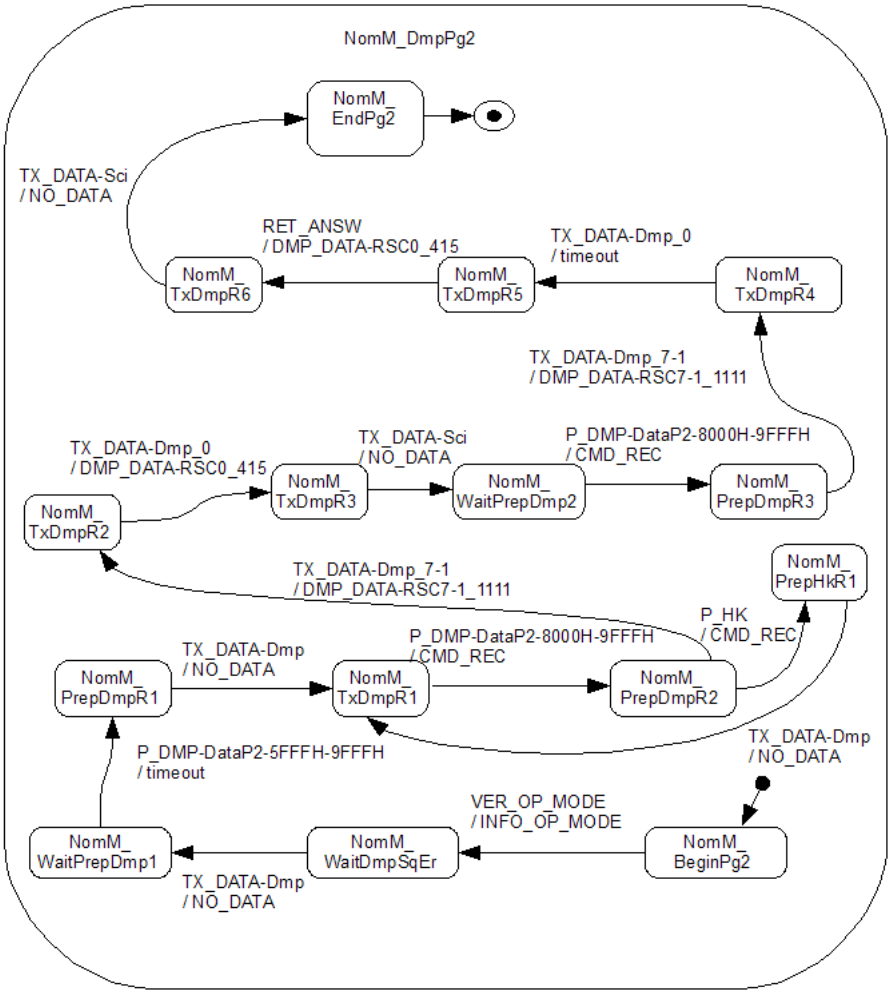

Figure 4.38 - Statechart Model: Scenario 14 (second hierarchy level).

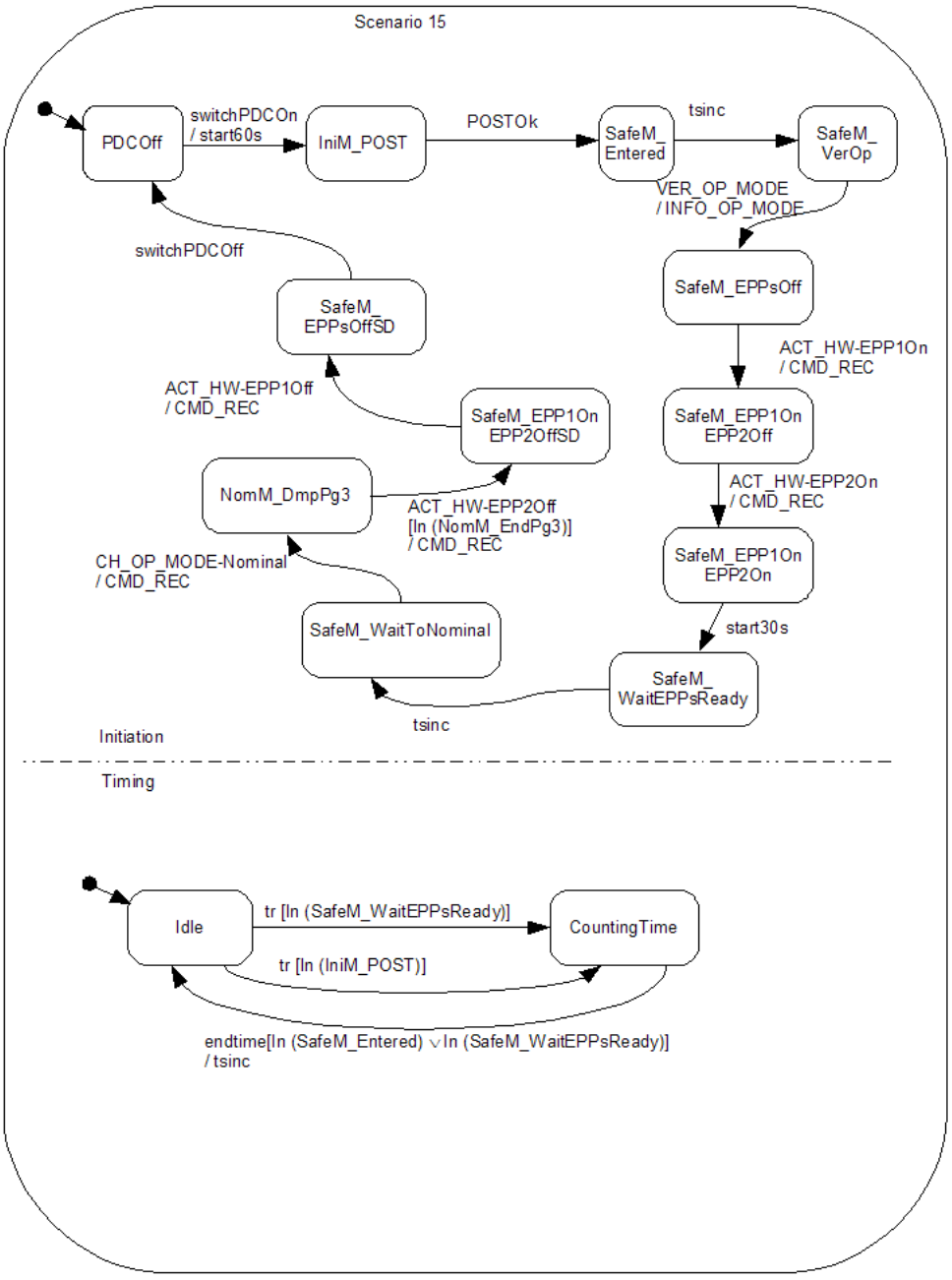

Figure 4.39 - Statechart Model: Scenario 15 (main model).

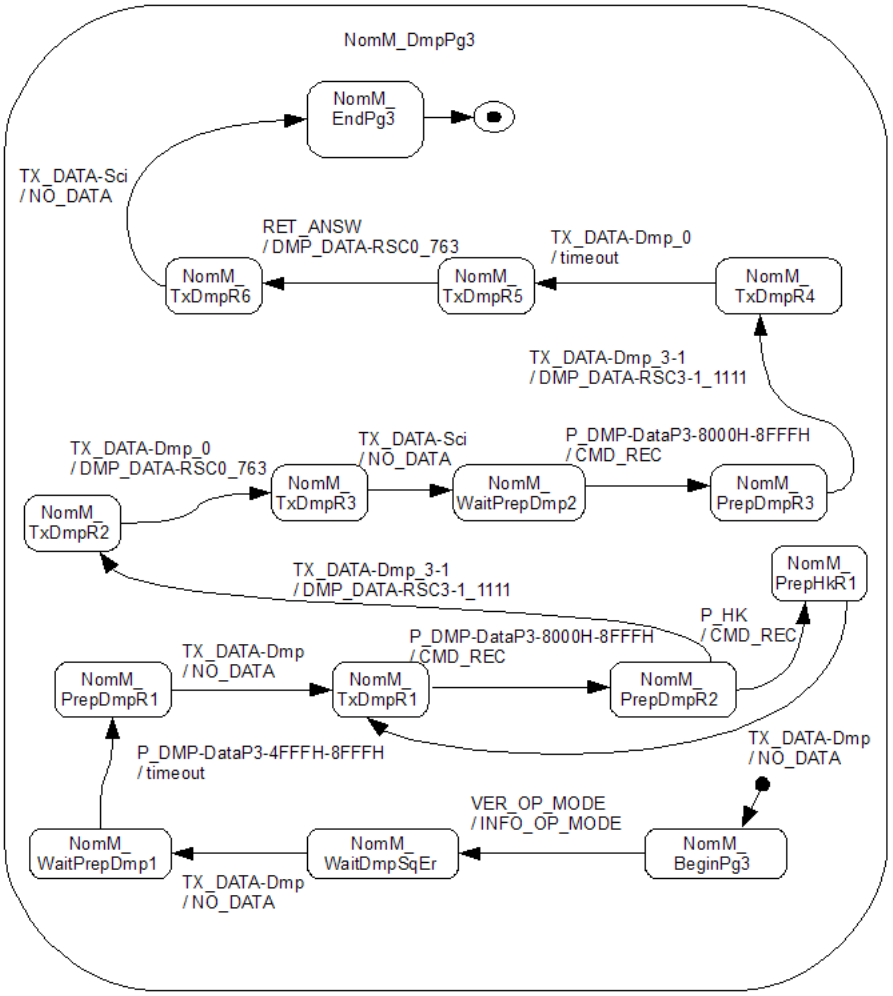

Figure 4.40 - Statechart Model: Scenario 15 (second hierarchy level).

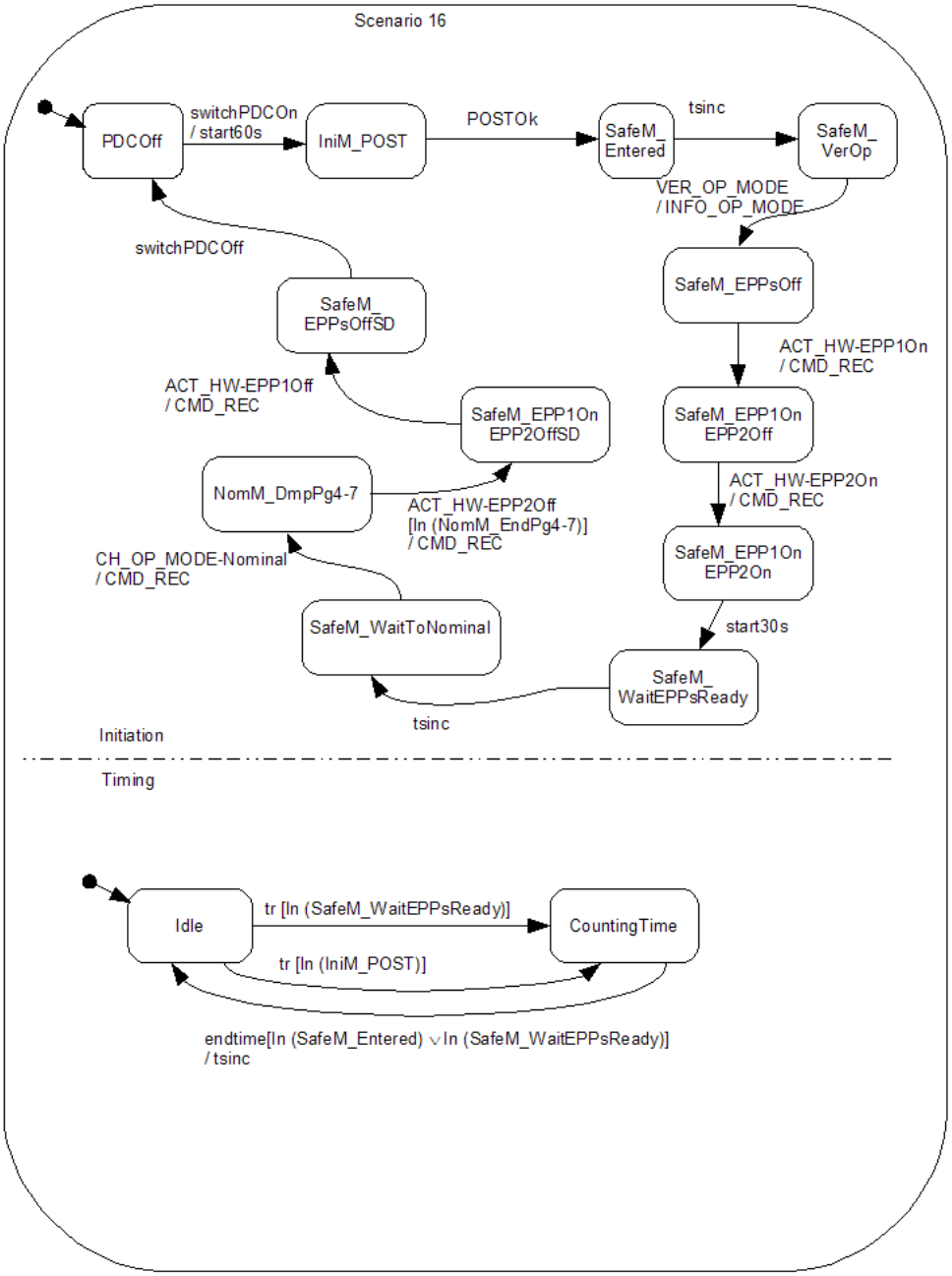

Figure 4.41 - Statechart Model: Scenario 16 (main model).

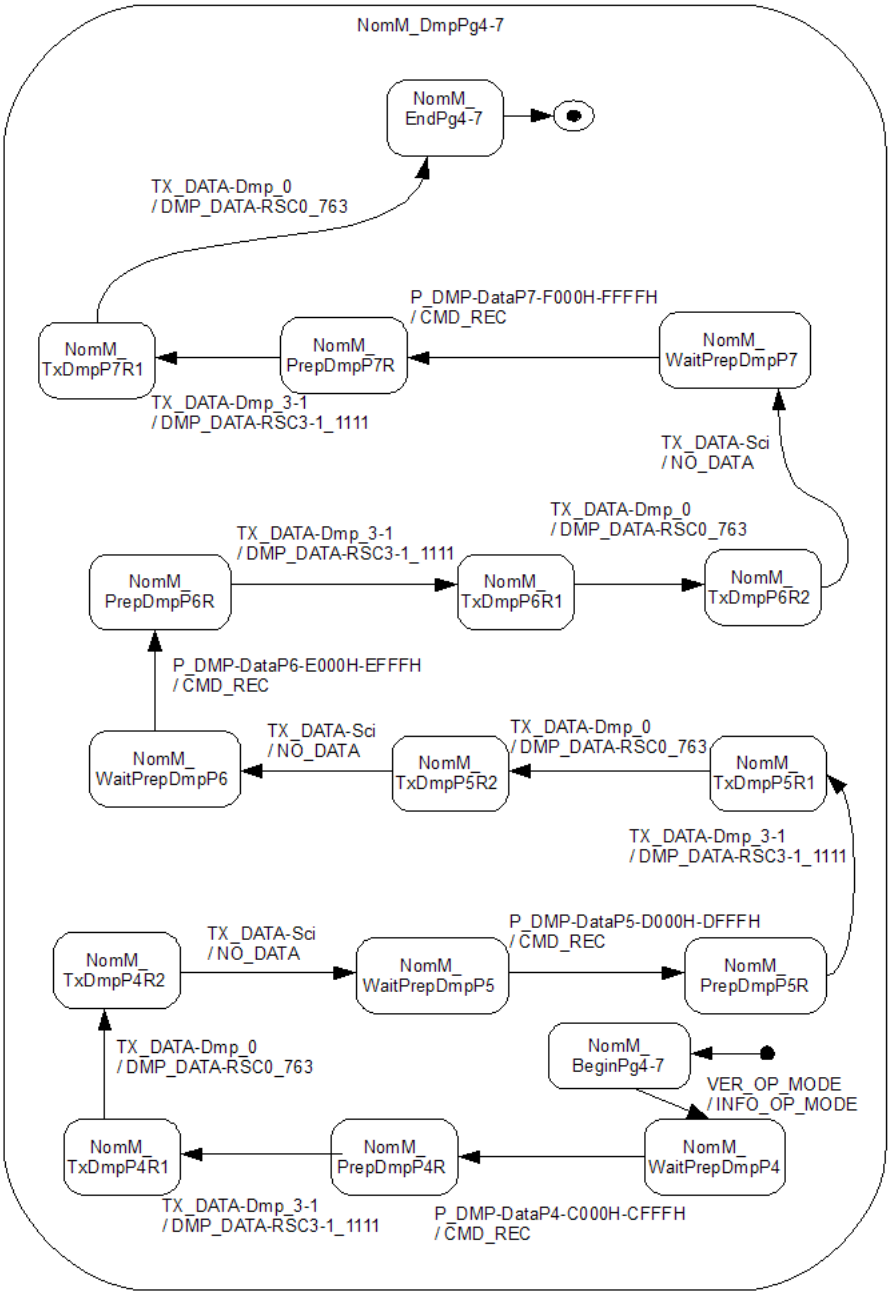

Figure 4.42 - Statechart Model: Scenario 16 (second hierarchy level).

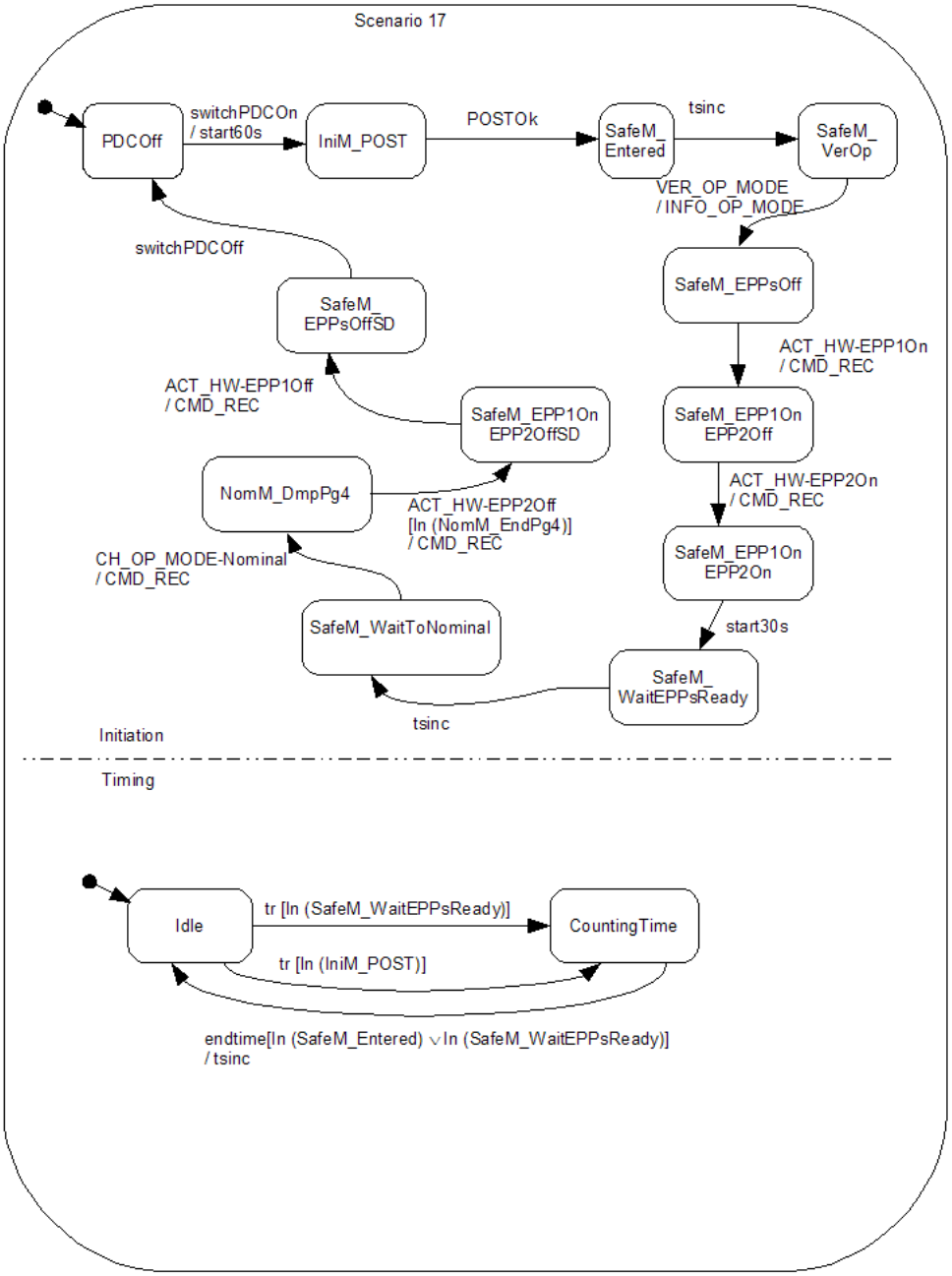

Figure 4.43 - Statechart Model: Scenario 17 (main model).

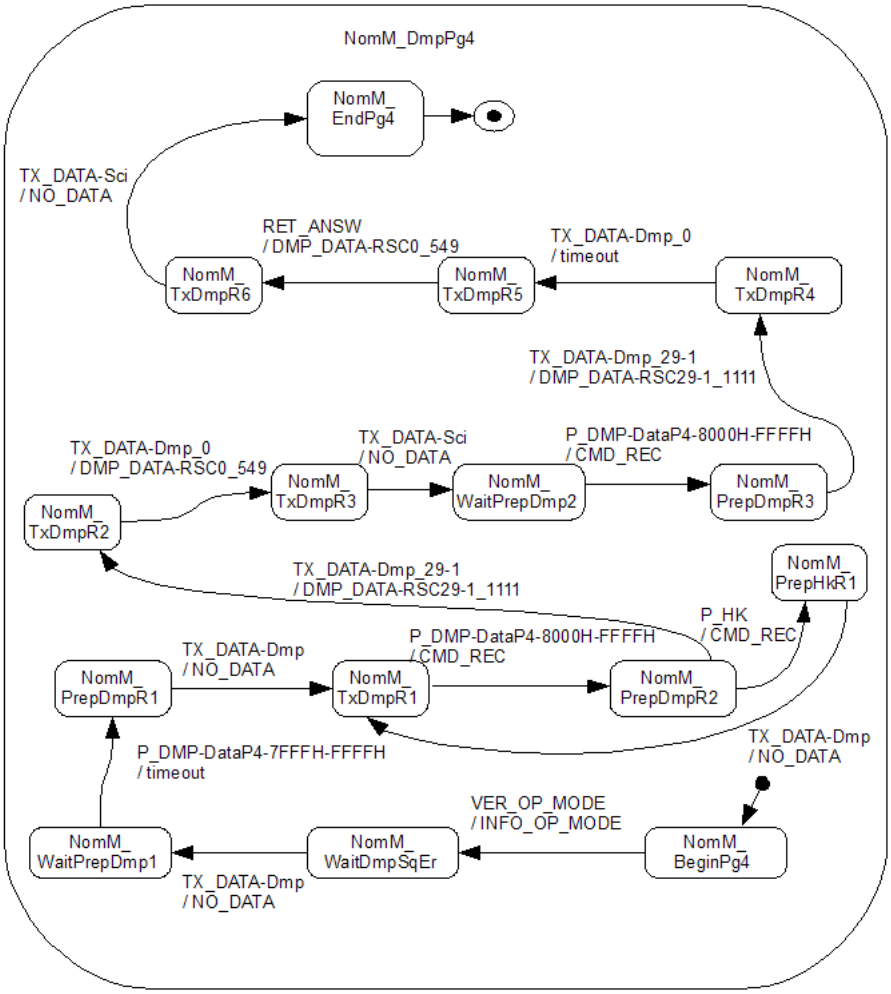

Figure 4.44 - Statechart Model: Scenario 17 (second hierarchy level).

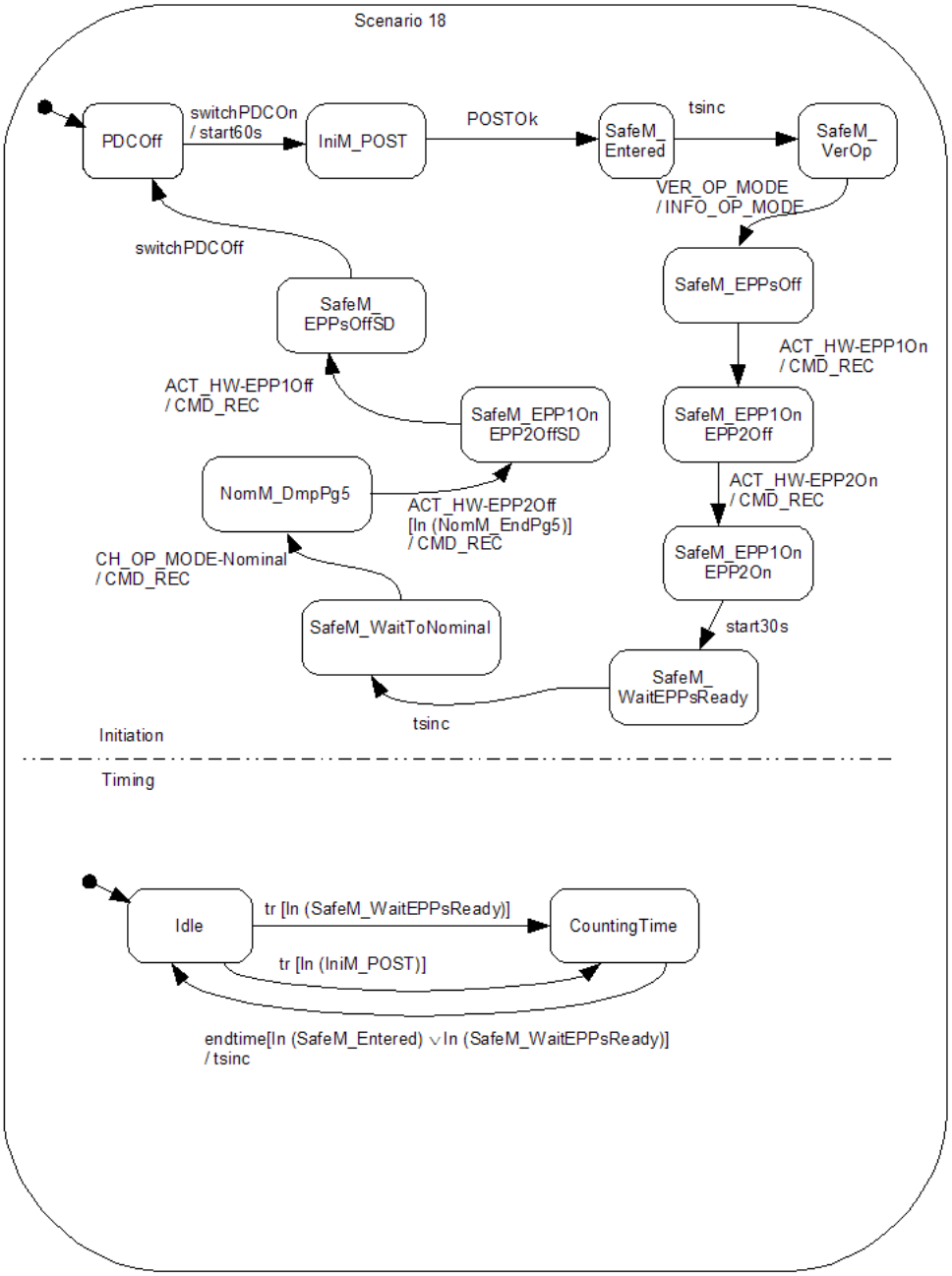

Figure 4.45 - Statechart Model: Scenario 18 (main model).

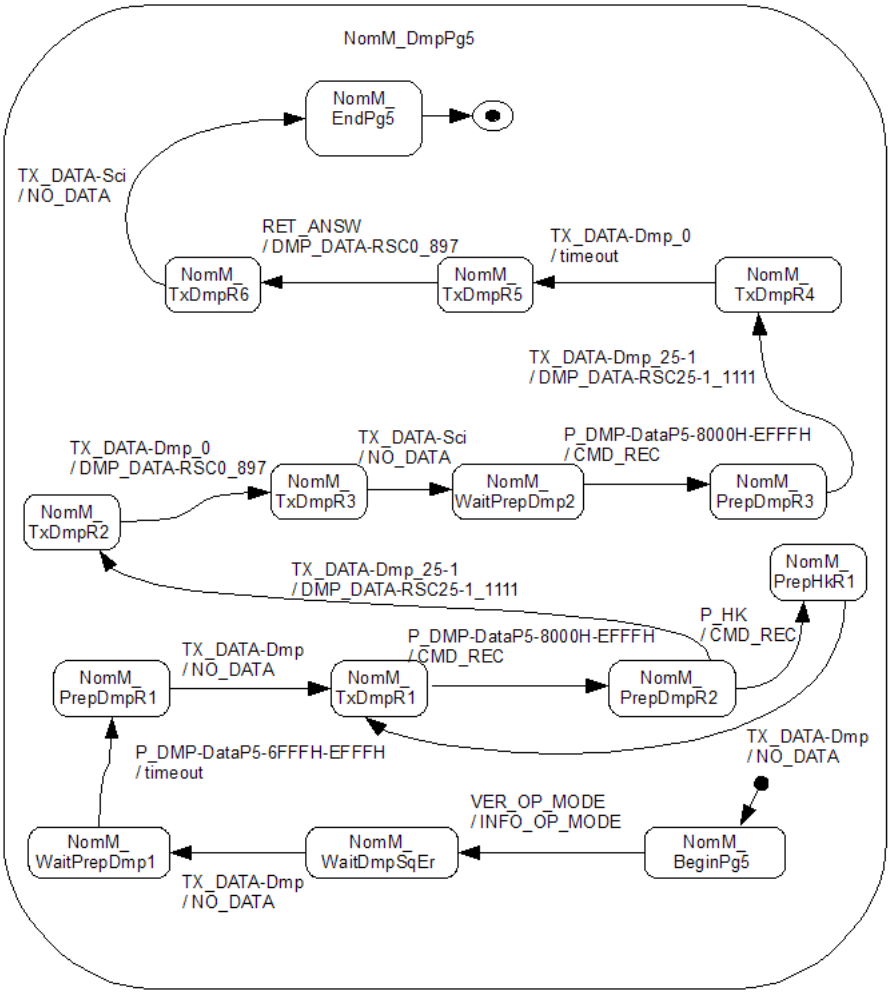

Figure 4.46 - Statechart Model: Scenario 18 (second hierarchy level).

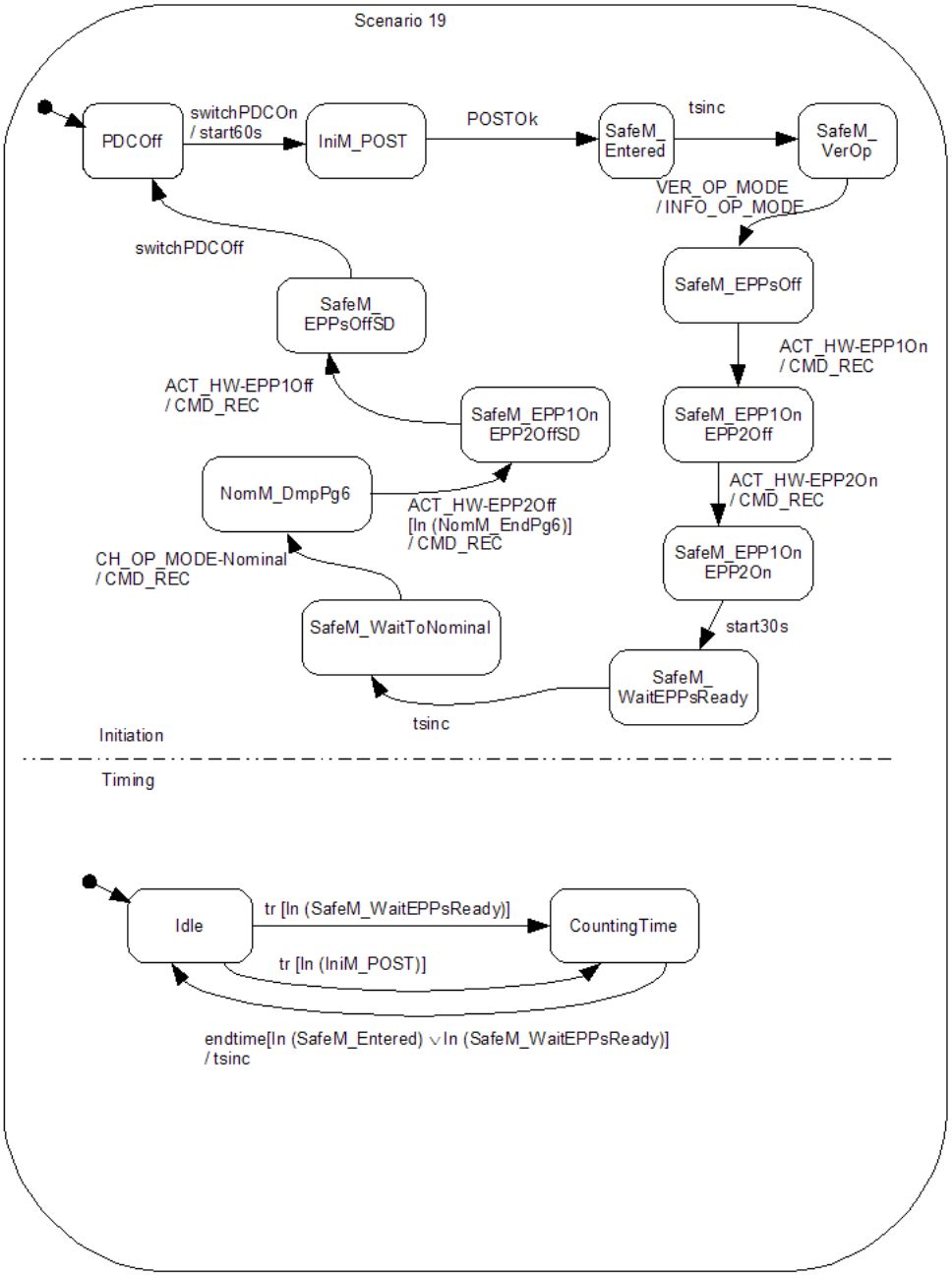

Figure 4.47 - Statechart Model: Scenario 19 (main model).

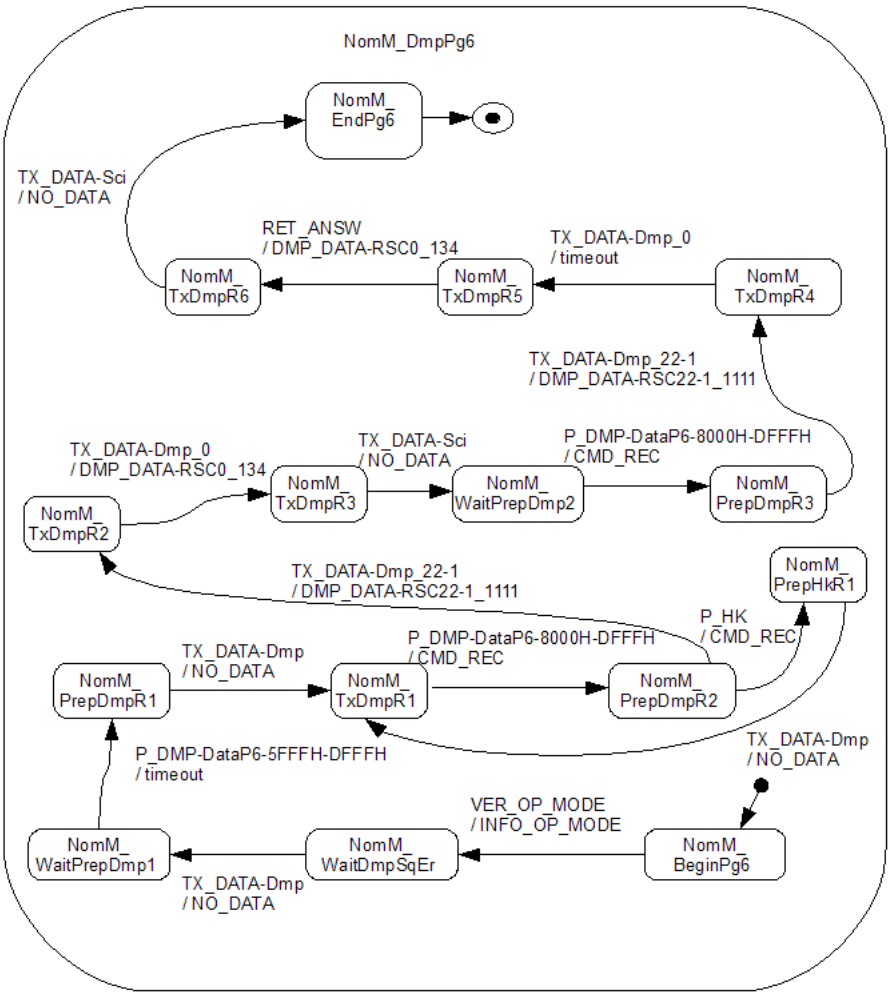

Figure 4.48 - Statechart Model: Scenario 19 (second hierarchy level).

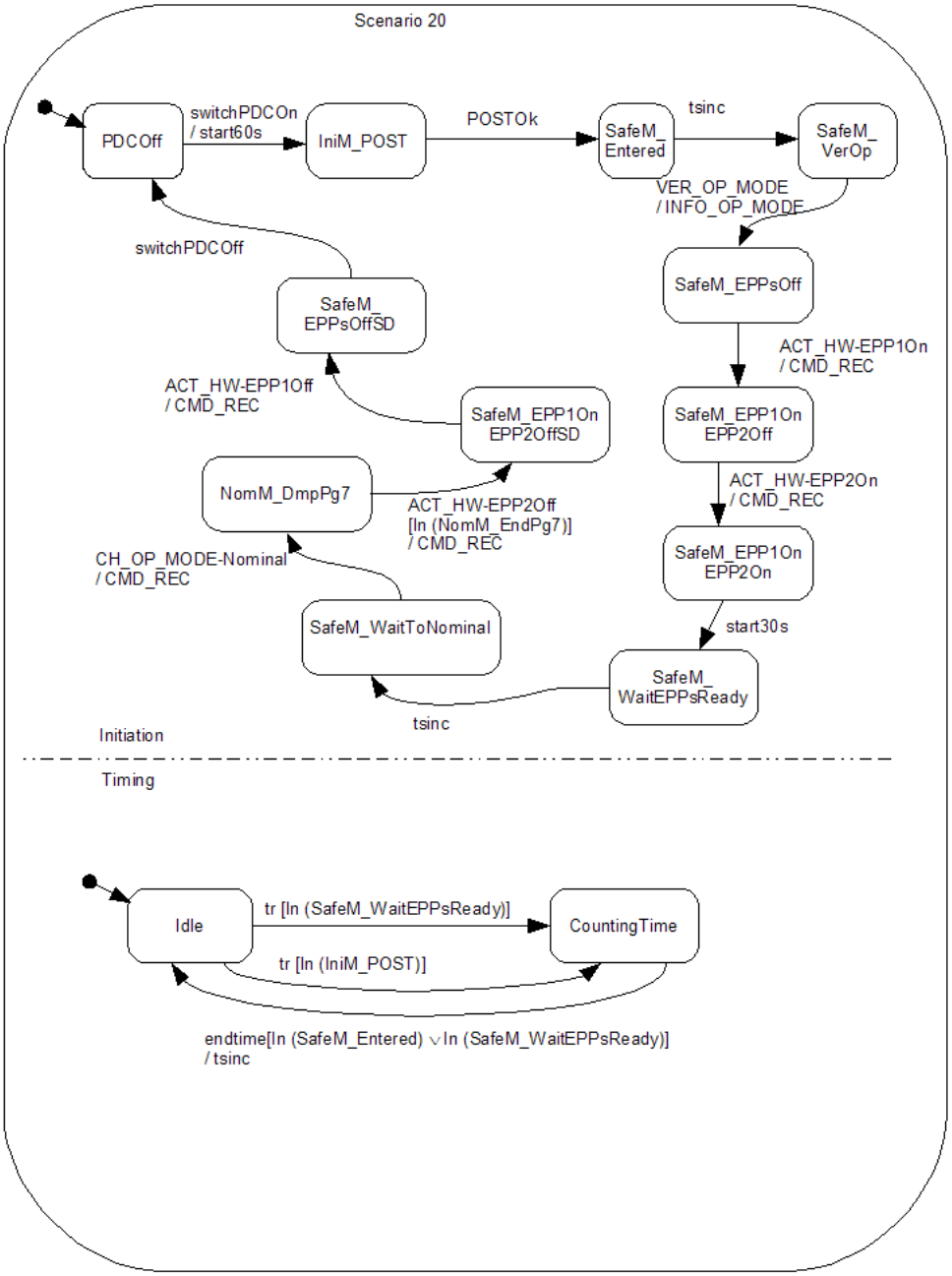

Figure 4.49 - Statechart Model: Scenario 20 (main model).

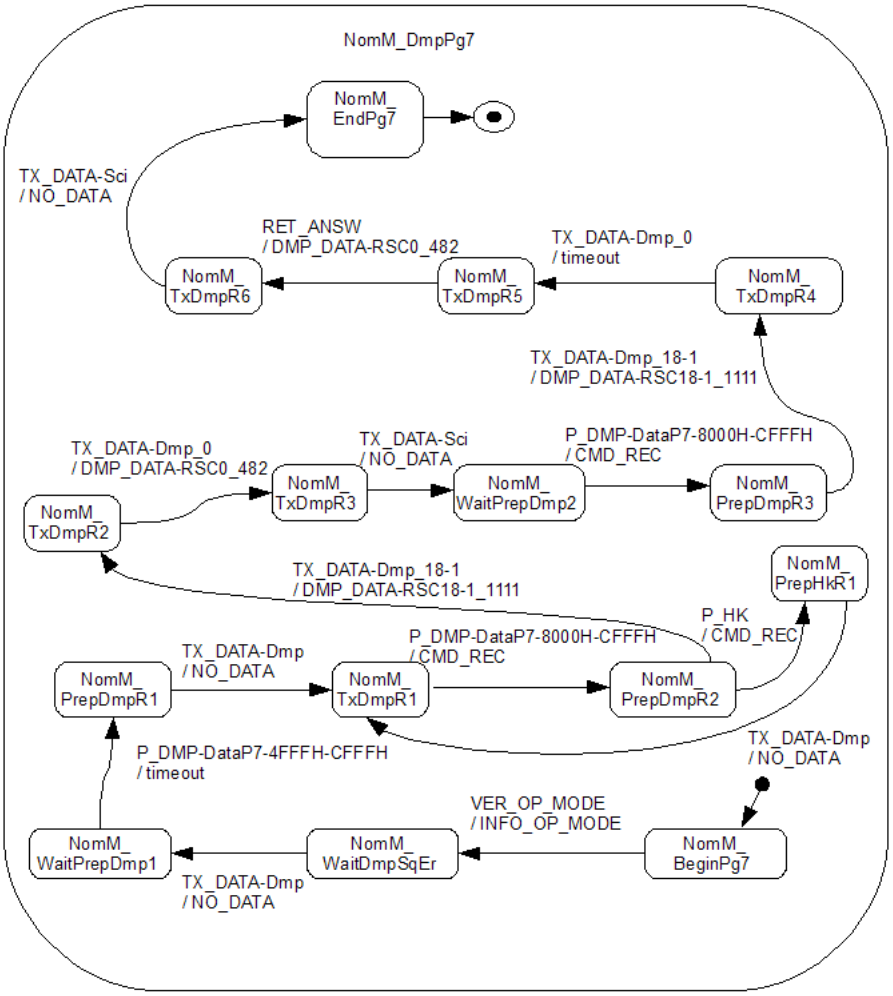

Figure 4.50 - Statechart Model: Scenario 20 (second hierarchy level).

4.4 Summary

After applying each technique to the preceding problems, the first and quite obvious conclusion was that all looked very different: models, test cases and methods or tactics were, initially, hard to compare. Hence, a framework was necessary in order to compare both techniques and tools. After searching for articles comparing MBT techniques and tools (PARADKAR, 2005; NETO et al., 2008; SINHA et al., 2006), the work of Paradkar (PARADKAR, 2005) was followed and the set of dimensions defined by the author to base the comparison was extended. These dimensions were divided into two groups summarized in Tables 4.4 and 4.5. Table 4.4 includes those dimensions that are, at least to some extent, independent of the cases studies, while Table 4.5 reunites those dimensions whose values depend on each case study.

The dimensions named **Notation concepts** and **MBT concepts** include the new concepts a software engineer with a basic knowledge on formal modelling has to learn in order to apply each technique. The dimensions whose values are marked with **X** indicate which technique is more suitable for each dimension –this does not necessarily mean that the other technique cannot be applied at all. In column **No. of test cases** of Table 4.5, it is shown for GTSC the number of test cases and (/) the number of test steps since "test step" is a meaningful concept regarding FSM-based MBT techniques. On the other hand, for Fastest it is given the number of test cases since each test case has always only one test step. The dimension called **Ratio** is the ratio between the two left rows. By indicating the **Model size** the goal was to give a broad idea of the model complexity. For the Z models it is provided the number of Z-LaTeX lines of code, while for the Statecharts models it is shown the number of states and (/) the number of transitions of the flat FSM. Although these measures are incomparable to each other, they give an idea of the relative complexity: (a) between the case studies, and (b) with respect to other models that can be found in the literature.

Table 4.4 shows that both techniques are rather similar, although GTSC can be applied to more phases of the testing process. As easily noted, Table 4.5 shows that GTSC outperforms Fastest when reactive systems are considered (EXP and SWPDC), but Fastest wins when the tools are applied to a (more) information oriented system (Scheduler).

Although this comparison was useful, it was realized a possibility for complementing

Table 4.4 - Comparison criteria. Case-study-independent dimensions.

Dimension	**GTSC** – INPE	**Fastest** – CIFASIS & Flowgate
Notation concepts	Statecharts: XOR and AND states, external events, actions (internal events or outputs), conditions, hierarchy, parallelism and machine synchronization, broadcasting, shallow and deep history, PcML language	Z (subset): first order logic, state and operation schema, schema language, typed logic, typed set theory, before and after state convention, input and output variables convention, logic and mathematics for modelling
MBT concepts	reachability tree, state configuration, Statecharts flattening, determinism, test criteria for FSM (switch cover, UIO and DS) and for Statecharts (all-transitions and all-simple-paths)	testing tactics (standard partitions, disjunctive normal form, free types), testing tree, domain partition, valid input space
Computability issues	state explosion	huge finite models
Computability solutions	duplicate nodes (Statecharts test case generation)	parallelization, user assistance
Test objectives	usage scenarios and models to represent them	written as Z schema boxes
Test cases	sequences of events producing state transitions	bindings between state or input variables, and constant values
Unit testing		**X**
Integration testing	**X**	
System testing	**X**	
Acceptance testing	**X**	
Reactive systems	**X**	
Information systems		**X**

Table 4.5 - Comparison criteria. Case-study-dependent dimensions. Times are in minutes.

Case study	Dimensions							
	Comp. time		No. of test cases		Ratio		Model size	
	GTSC	Fastest	GTSC	Fastest	GTSC	Fastest	GTSC	Fastest
EXP	0:51	124:00	78/436	112	0:0.6	1:06	16/61	608
Scheduler	3:20	3:00	5/43	29	0:40	0:06	7/16	240
SWPDC	11:42	158:00	30/663	117	0:23	1:21	512/522	1,238

both techniques, as is shown in the next chapter.

5 A COMPLEMENTARY APPROACH

As seen in Table 4.4 test cases generated by GTSC are sequences of state transitions, while test cases generated by Fastest are bindings or assignments between state variables and constant values. Figure 5.1 shows two test cases generated by GTSC for the EXP case study.

$$0xEB \rightarrow 0x92 \rightarrow T1B \rightarrow Size38\_E \rightarrow Data0x100\_E \rightarrow TimerTimeout$$

$$0xEB \rightarrow 0x92 \rightarrow T1B \rightarrow Size38\_E \rightarrow Data0x100\_E \rightarrow TypeOkSizeOkDataOk\_E \rightarrow Cks\_Ok$$

Figure 5.1 - Two test cases generated by GTSC for the EXP case study.

In Figure 5.1, the test cases correspond to OBDH sending a memory load command to EXP –i.e. OBDH sends a program to EXP that it must load into its computer memory. $0xEB$ and $0x92$ identify the beginning of a new order coming from OBDH. $T1B$ means that the sequences correspond to a memory load command. $Size38\_E$ and $Data0x100\_E$ are the size in Bytes of a piece of program, and the address in memory where this piece of code will be loaded followed by the code, respectively. The event $TypeOkSizeOkDataOK\_E$ means the data (code) sent by OBDH was received correctly by EXP. Note that the first test sequence ends in a timeout event. This means that OBDH has taken too long in sending all the data, and

so actually no memory load is performed. The second test sequence describes a successful command, i.e. EXP loads the code into its computer memory.

Figure 5.2 shows Z schema boxes which are two test cases generated by Fastest for the same memory load command.

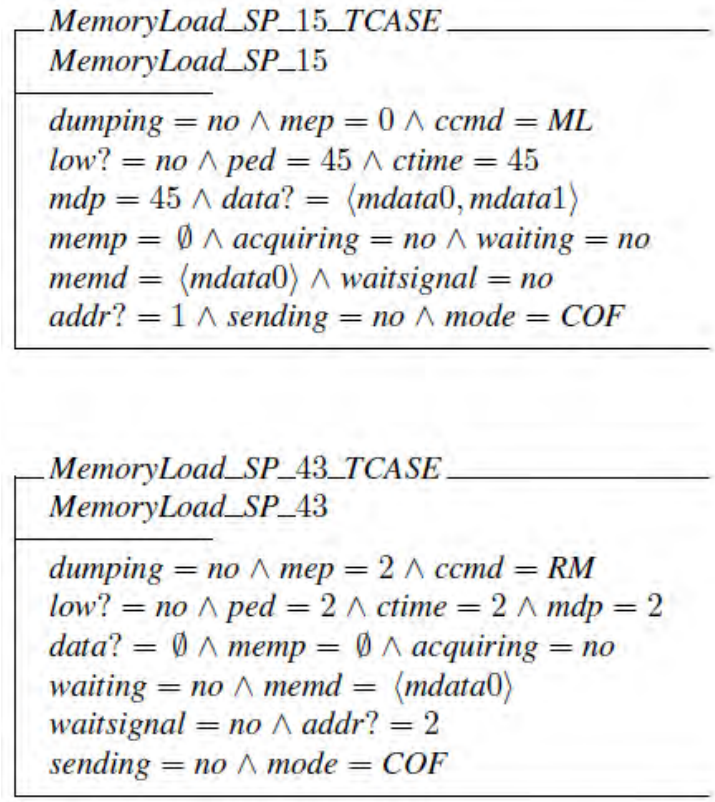

Figure 5.2 - Z schema boxes for the memory load command.

Observe that each test case is a set of assignments[1]. Each assignment sets a constant value for one of the state and input variables. Regarding this command, the most important variables are *data*?, which is the input variable representing the new program; *addr*?, representing the initial address where the program must be loaded; and *memd*, which is the state variable representing EXP's computer memory. Then, roughly speaking, the first test case says that OBDH has sent a two byte long program to be loaded starting at the first memory location while EXP's memory holds a one byte long program; and, on the other hand, the second test case says

[1]Actually they are not assignments but propositional equalities.

that OBDH has sent an empty program –that should be loaded after the program currently being held by EXP– while EXP's computer memory is occupied by a single byte program.

After arriving at this point, an approach that combines both techniques emerged quite easily. All the test sequences generated by GTSC that include in the execution of a data intensive, underspecified[2] operation –with respect to the Statecharts model– are considered and, at that point, all the test cases generated by Fastest for that operation are executed. For example, since the first GTSC test sequence written above does not include in the execution of a data intensive operation, then any Z test case is executed for it; however, 28 Z test cases are executed for the second test sequence since it represents EXP loading its memory with a new program. In this way, the software is tested as a whole and every time it gets to a point where a complex data-intensive operation has to be executed, then this operation is fully tested too.

The approach is now presented in a more formal and systematic way. Let E be the set of events in the Statecharts model and D the subset of E that represents the execution of some data-intensive operation, which have also been specified as Z operations. Let S be the set of test sequences derived from the Statecharts model. Now, consider the subset of S, S_D, including all the test sequences that contain some element of D. For each d in D, define T_d as the set of tests derived from the Z model for unit d; each t in T_d is a call to d with some specific input parameters. If P is the program that has been modeled in Statecharts and Z, then test P with the set of test sequences as defined in Figure 5.3.

$$S_T = \{s_i \frown \langle t \rangle \frown s_f \mid (\exists d \in D : s_i \frown \langle d \rangle \frown s_f \in S_D \wedge t \in T_d\}) \cup (S \setminus S_D)$$

Figure 5.3 - Formalization of the combined approach.

[2]Underspecification in Statecharts models comes into picture due to two main reasons. First, Statecharts are based on FSM, and Z on first order logic. First order logic is more expressive than FSM, regarding data transformation operations. Second, the ability (or lack of ability) of the test designer in adequately modeling the IUT for testing. And this lack of ability occurs probably because a test designer relies the development of the models based on documents (requirements specifications, ...) which present serious issues. In some cases, there is no documentation at all available which is a worse situation.

In other words, any test sequence must be broken in the points where there is a call to a data-intensive operation and replace these points with each test case generated from the Z model for the corresponding operation.

As this formalization shows, the method can be made automatic by simply using a convenient naming convention.

Coming back to the EXP-OBDH case study, there are two data intensive operations: memory load and transmit data. Fastest yielded 28 and 20 test cases, respectively, for these operations. On the other hand, GTSC generates two test sequences including the execution of these operations. Then, the original 78 test cases generated by GTSC become 126, improving the original test suite in about 60%. Furthermore, there is not only an improvement in the number of test cases but also (a) in the chances to uncover defects (faults) in complex operations that would not be thoroughly tested by any method alone; and (b) in the fact that the test cases provided by Fastest give the exact input constants that must be provided to the IUT, thus augmenting the proportion of automatic steps of the testing process.

The research methodology that was developed before getting these results made the Z models unnecessarily complex because they describe not only those operations underspecified in the Statecharts model but also all the simpler and reactive ones. With this combined approach, the Z model must describe only data intensive operations. In this way both the effort and the size (complexity) of the Z model can be reduced. For instance, the EXP-OBDH Z model is 608 ZLOC (Table 4.5) but only less than 241 are involved in the data intensive operations.

It is worth to be mentioned that Statecharts can be used to model data-intensive systems, and, on the other hand, Z can be used to model reactive systems. Hence it would be possible to use only one approach, but it would be easier and more intuitive to combine both approaches as described in this chapter.

6 CONCLUSIONS

In this Technical Report, two MBT techniques and tools were compared and an approach was presented in order to combine those techniques to improve the testing of systems including both reactive and data intensive functions. The main conclusions of this work so far are: (a) it is rather easy to combine the approaches,

and (b) that the combined approach could potentially uncover many defects that each technique alone cannot.

There is a lot of work to do in the future. First, it is necessary to assess the amount of defects uncovered by the combined approach by applying, for instance, mutation analysis. In doing so, a comparison between the new combined methodology and the previous individual methodologies developed by each team can be done. Second, new case studies shall be addressed so that what have learned so far can be applied– for example, writing shorter Z models that efficiently complement the Statecharts model. Third, both tools will be integrated resulting in a more automatic process.

REFERENCES

ABDURAZIK, A.; OFFUTT, J. Using UML collaboration diagrams for static checking and test generation. In: **UML 2000 - the Unified Modeling Language**. Berlin/Heidelberg, Germany: Springer Berlin/Heidelberg, 2000. v. 1939, p. 383–395. Lecture notes in computer science. 13

AMBROSIO, A. M.; MATTIELLO-FRANCISCO, F.; SANTIAGO, V. A.; SILVA, W. P.; MARTINS, E. Designing fault injection experiments using state-based model to test a space software. In: **Dependable Computing**. Berlin/Heidelberg, Germany: Springer Berlin/Heidelberg, 2007. v. 4746, p. 170–178. Lecture notes in computer science. 9

ARANTES, A. O.; VIJAYKUMAR, N. L.; SANTIAGO, V. A.; GUIMARAES, D. WEB-PerformCharts: a collaborative web-based tool for test case generation from Statecharts. In: INTERNATIONAL CONFERENCE ON INFORMATION INTEGRATION AND WEB-BASED APPLICATIONS & SERVICES (IIWAS), 10., 2008, Linz, Austria. **Proceedings...** New York, NY, USA: ACM, 2008. p. 374–381. 11

BINDER, R. V. **Testing object-oriented systems**: models, patterns, and tools. USA: Addison-Wesley Professional, 1999. 1248 p. 9

COMMUNITY Z TOOLS. 2009. Available from: <http://czt.sourceforge.net/>. Access in: Oct 20, 2009. 11

CRISTIÁ, M.; MONETTI, P. R. Implementing and applying the Stocks-Carrington framework for model-based testing. In: **Formal Methods in Software Engineering**. Berlin/Heidelberg, Germany: Springer Berlin/Heidelberg, 2009. v. 5885, p. 167–185. Lecture notes in computer science. 11

HIERONS, R. M.; BOGDANOV, K.; BOWEN, J. P.; CLEAVELAND, R.; DERRICK, J.; DICK, J.; GHEORGHE, M.; HARMAN, M.; KAPOOR, K.; KRAUSE, P.; LüTTGEN, G.; SIMONS, A. J. H.; VILKOMIR, S.; WOODWARD, M. R.; ZEDAN, H. Using formal specifications to support testing. **ACM Computing Surveys**, v. 41, n. 2, p. 1–76, 2009. 8

HIERONS, R. M.; SADEGHIPOUR, S.; SINGH, H. Testing a system specified using Statecharts and Z. **Information and Software Technology**, v. 43, n. 2, p. 137–149, 2001. 8

MACCOLL, I.; CARRINGTON, D. Extending the Test Template Framework. In: BCS-FACS NORTHERN FORMAL METHODS WORKSHOP, 3., 1998, Ilkley, UK. **Proceedings...** [S.l.], 1998. p. 1–17. 11

MASIERO, P. C.; MALDONADO, J. C.; BOAVENTURA, I. G. A reachability tree for Statecharts and analysis of some properties. **Information and Software Technology**, v. 36, n. 10, p. 615–624, 1994. 10

MATHUR, A. P. **Foundations of software testing**. Delhi, India: Dorling Kindersley (India), Pearson Education in South Asia, 2008. 689 p. 16, 19

NETO, A. D.; SUBRAMANYAN, R.; VIEIRA, M.; TRAVASSOS, G. H.; SHULL, F. Improving evidence about software technologies: a look at model-based testing. **IEEE Software**, v. 25, n. 3, p. 10–13, 2008. 8, 60

PARADKAR, A. Case studies on fault detection effectiveness of model based test generation techniques. In: WORKSHOP ON ADVANCES IN MODEL-BASED TESTING (A-MOST), 1., 2005, St. Louis, MO, USA. **Proceedings...** New York, NY, USA: ACM, 2005. p. 1–7. 8, 60

PETRENKO, A.; YEVTUSHENKO, N. Testing from partial deterministic FSM specifications. **IEEE Transactions on Computers**, v. 54, n. 9, p. 1154–1165, 2005. 9

PIMONT, S.; RAULT, J. C. A software reliability assessment based on a structural and behavioral analysis of programs. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (ICSE), 2., 1976, San Francisco, CA, USA. **Proceedings...** New York, NY, USA: ACM, 1976. p. 486–491. 9

SANTIAGO, V.; AMARAL, A. S. M.; VIJAYKUMAR, N. L.; MATTIELLO-FRANCISCO, M. F.; MARTINS, E.; LOPES, O. C. A practical approach for automated test case generation using Statecharts. In: ANNUAL INTERNATIONAL COMPUTER SOFTWARE & APPLICATIONS CONFERENCE (COMPSAC) - INTERNATIONAL WORKSHOP ON TESTING AND QUALITY ASSURANCE FOR COMPONENT-BASED SYSTEMS

(TQACBS), 30., 2006, Chicago, IL, USA. **Proceedings...** Los Alamitos, CA, USA: IEEE Computer Society, 2006. p. 183–188. 9

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 & KNOWLEDGE ENGINEERING (SEKE), 19., 2007, Boston, MA, USA. **Proceedings...** Skokie, IL, USA: Knowledge Systems Institute Graduate School, 2007. p. 51–56. 19

SANTIAGO, V.; VIJAYKUMAR, N. L.; GUIMARAES, 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) - WORKSHOP ON ADVANCES IN MODEL BASED TESTING (A-MOST), 1., 2008, Lillehammer, Norway. **Proceedings...** Washington, DC, USA: IEEE Computer Society, 2008. p. 63–72. 1 CD-ROM. 9, 10, 11

SIDHU, D. P.; LEUNG, T. K. Formal methods for protocol testing: a detailed study. **IEEE Transactions on Software Engineering**, v. 15, n. 4, p. 413–426, 1989. 9

SINHA, A.; WILLIAMS, C. E.; SANTHANAM, P. A measurement framework for evaluating model-based test generation tools. **IBM Systems Journal**, v. 45, n. 3, p. 501–514, 2006. 60

SOUZA, S. R. S. **Validação de especificações de sistemas reativos: definição e análise de critérios de teste**. 264 p. Thesis (PhD in Applied Physics) — Universidade de São Paulo, São Carlos, SP, Brazil, 2000. 10

SPIVEY, J. M. **The Z notation**: a reference manual. Upper Saddle River, NJ, USA: Prentice-Hall, 1989. 155 p. 11

STOCKS, P.; CARRINGTON, D. A framework for specification-based testing. **IEEE Transactions on Software Engineering**, v. 22, n. 11, p. 777–793, 1996. 11

STOCKS, P. A. **Applying formal methods to software testing**. Thesis (PhD in Computer Science) — University of Queensland, Brisbane, Australia, 1994. 11

UTTING, M.; LEGEARD, B. **Practical model-based testing**: a tools approach. San Francisco, CA, USA: Morgan Kaufmann Publishers, 2007. 456 p. 15