Software Qual J manuscript No.
(will be inserted by the editor)

Generating Model-Based Test Cases from Natural Language Requirements for Space Application Software

Valdivino Alexandre de Santiago Júnior ·
Nandamudi Lankalapalli Vijaykumar

Received: date / Accepted: date

Abstract Natural Language (NL) deliverables suffer from ambiguity, poor understandability, incompleteness, and inconsistency. However, NL is straightforward and stakeholders are familiar with it to produce their software requirements documents. This paper presents a methodology, SOLIMVA, which aims at model-based test case generation considering NL requirements deliverables. The methodology is supported by a tool that makes it possible to automatically translate NL requirements into Statecharts models. Once the Statecharts are derived, another tool, GTSC, is used to generate the test cases. SOLIMVA uses combinatorial designs to identify scenarios for system and acceptance testing, and it requires that a test designer defines the application domain by means of a dictionary. Within the dictionary there is a *Semantic Translation Model* in which, among other features, a word sense disambiguation method helps in the translation process. Using as case study a space application software product, we compared SOLIMVA with a previous manual approach developed by an expert under two aspects: test objectives coverage and characteristics of the Executable Test Cases. In the first aspect, the SOLIMVA methodology not only covered the test objectives associated to the expert's scenarios but also proposed a better strategy with test objectives clearly separated according to the directives of combinatorial designs. The Executable Test Cases derived in accordance with the SOLIMVA methodology not only possessed similar characteristics with the expert's Executable Test Cases but also predicted behaviors that did not exist in the expert's strategy. The key benefits from applying the SOLIMVA methodology/tool within a Verification and Validation process are the easiness of use and, at the same time, the support of a formal method consequently leading to a potential acceptance of the methodology in complex software projects.

Valdivino Alexandre de Santiago Júnior
Instituto Nacional de Pesquisas Espaciais (INPE)
Av. dos Astronautas, 1758 – 12227-010
São José dos Campos – SP – Brazil
Tel.: +55-12-3208-7166
Fax: +55-12-3208-1875
E-mail: valdivino@das.inpe.br

Nandamudi Lankalapalli Vijaykumar
Instituto Nacional de Pesquisas Espaciais (INPE)
Av. dos Astronautas, 1758 – 12227-010
São José dos Campos – SP – Brazil

Keywords Model-Based Testing · Natural Language Requirements · Semantic Translation Model · Word Sense Disambiguation · Statecharts

1 Introduction

Verification and Validation (IEEE, 1990) encompass a wide array of Sofware Quality Assurance activities including formal technical reviews, quality and configuration audits, documentation review, feasibility study, and all sorts of testing (Pressman, 2001). Thus, testing a software product is only a facet to get quality. However, the role of testing is undoubtedly important and it has received attention from both industry and academia.

In system and acceptance testing, test cases/sequences are derived considering the entire software product. In this case, black box testing techniques (Mathur, 2008) are usually adopted. Besides, a scenario-based approach is also recommended for system and acceptance test case generation where distinct interactions with the system are addressed.

A black box testing technique, which is state of the art, is exactly **Model-Based Testing**. According to some authors, the testing community considers Model-Based Testing as deriving tests from software behavioral models (El-Far and Whittaker, 2001). Such consideration includes formal model/language specifications and other non-formal notations, like Unified Modeling Language (UML) models (OMG, 2007). Among the formal methods used for system and acceptance model-based test case generation are Statecharts (Harel, 1987; Santiago et al, 2008b), Finite State Machines (FSMs) (Sidhu and Leung, 1989), and Z (Spivey, 1989; Cristiá et al, 2010).

In model-based system and acceptance test case generation, a test designer usually breaks down the entire system based on functionalities it must provide (or interactions with the system in case of scenario-based approaches), and then models are derived to address each functionality. Based on such models, test cases can be obtained. However, identification of scenarios that consequently leads to test case generation is not an easy task and it is time consuming. Test designers try to identify scenarios based on the very first deliverables (artifacts) created within the software development lifecycle, such as software requirements specifications. Even though a development team might produce software requirements specifications using scenario-based methods, like use case models (OMG, 2007), a test designer must not rely only on the "perspective" of the developers because, in doing so, he/she will not have the independent point of view that is crucial in test case generation.

Software requirements specifications can be developed according to various approaches. Requirements may be elicited and modeled in accordance with scenario-based methods, such as use case models, and goal-oriented methods, such as Tropos (Bresciani et al, 2004). Formal methods (models, languages) may be used to represent behavior, but they require high expertise for that and they are difficult to integrate with the ordinary software develpment process adopted in industry. Thus, managers in industry usually avoid the inclusion of formal methods in their established processes (Abrial, 2006).

The conclusion is that Natural Language (NL) is still the most used to produce software requirements specifications (Mich et al, 2004; Abrial, 2006) as it is the simplest way for stakeholders. Moreover, NL may be associated to requirements modeling methods, like use case models where a textual description exists in order to narrate the behavior through a sequence of actor-system interactions. Thus, NL plays a significant role in use case requirements specifications because actors, actions, scenarios, etc. are described in NL (Fantechi et al, 2003).

Due to what was previously mentioned and in particular taking into account NL requirements documents, identification of scenarios, their respective models and test case generation are arduous and time-consuming tasks, especially for real complex applications such as software embedded in on-board computers of satellites. Therefore, this paper presents a methodology, SOLIMVA, which aims at model-based test case generation considering NL requirements deliverables. The methodology is supported by a tool that makes it possible to automatically translate NL requirements into Statecharts models. Once the Statecharts are derived, another tool, the *Geração Automática de Casos de Teste Baseada em Statecharts* (GTSC - Automated Test Case Generation based on Statecharts) environment (Santiago et al, 2008b), is used to generate the test cases. The SOLIMVA methodology relies on combinatorial designs (Mathur, 2008) to identify scenarios for system and acceptance testing. The tool that supports SOLIMVA uses computational linguistics techniques in order to reason about some semantic aspects of the model to be generated. Using as case study a space application software product, we compared SOLIMVA with a previous manual approach developed by an expert under two aspects: test objectives coverage and characteristics of the Executable Test Cases. In the first aspect, the SOLIMVA methodology not only covered the test objectives associated to the expert's scenarios but also proposed a better strategy with test objectives clearly separated according to the directives of combinatorial designs. The Executable Test Cases derived in accordance with the SOLIMVA methodology not only possessed similar characteristics with the expert's Executable Test Cases but also predicted behaviors that did not exist in the expert's strategy. The key benefits from applying the SOLIMVA methodology/tool within a Verification and Validation process are the easiness of use and, at the same time, the support of a formal method consequently leading to a potential acceptance of the methodology in complex software projects.

This paper is organized as follows. Section 2 presents the case study, a space application software product, in which the SOLIMVA methodology and its tool were applied. Section 3 presents the SOLIMVA methodology. Section 4 details the activity within the methodology that requires significant computational effort to be executed. Section 5 shows the application of the SOLIMVA methodology/tool to the case study described in Section 2. Section 6 presents general remarks about this research. Other proposals in the literature as well as a comparison among such proposals and the SOLIMVA methodology are presented in Section 7. Conclusions and future directions are in Section 8.

2 Case study: software embedded in satellite payload

In order to improve understandability, we are first going to describe the case study in which the SOLIMVA methodology and its supporting tool were applied. This case study is a space application software product, Software for the Payload Data Handling Computer (SWPDC), developed in the context of the *Qualidade do Software Embarcado em Aplicações Espaciais* (QSEE - Quality of Space Application Embedded Software) research project (Santiago et al, 2007). QSEE was an experience in outsourcing the development of software embedded in satellite payload. The *Instituto Nacional de Pesquisas Espaciais* (INPE - National Institute for Space Research) was the customer and there were two SWPDC's suppliers: INPE itself and a Brazilian software company. The QSEE research project used the European Cooperation for Space Standardization (ECSS) standards (ECSS, 2008) in order to guide the relationship between customer and supplier.

Fig. 1 shows the Computing Subsystem Functional Architecture defined for QSEE project. Note the following computing units in the architecture: On-Board Data Handling (OBDH) Computer, Payload Data Handling Computer (PDC), Event Pre-Processors (EPPs; EPP H1 and EPP H2),

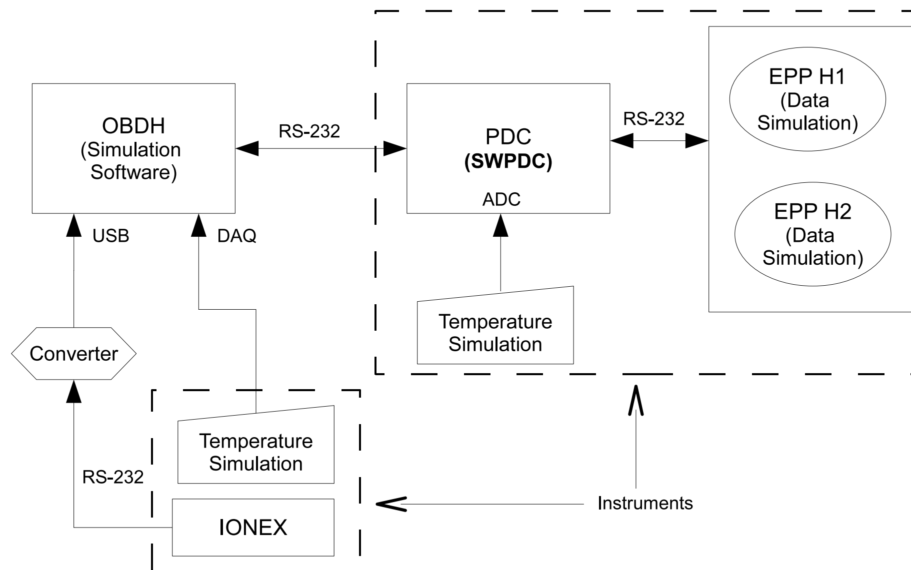


Fig. 1 Computing Subsystem Functional Architecture defined for QSEE project. Caption: ADC = Analog-to-Digital Converter; DAQ = Data Acquisition Board; RS-232 = Recommended Standard 232; USB = Universal Serial Bus. Source: adapted from Santiago et al (2007)

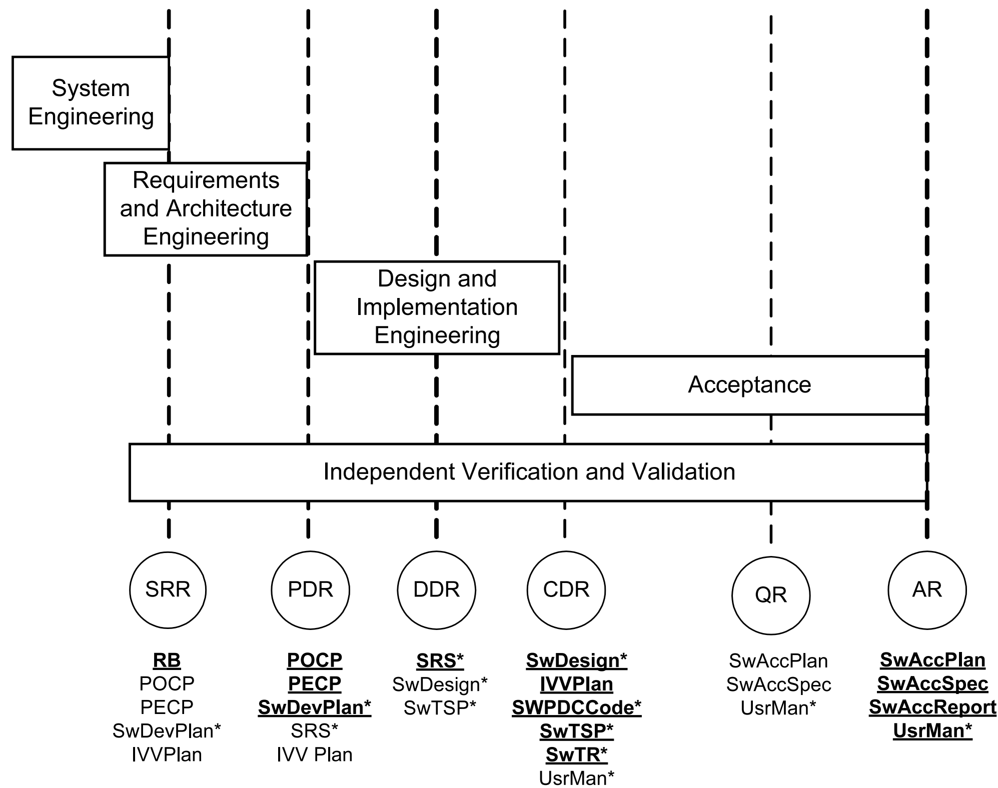
and Ionospheric Plasma Experiments (IONEX) Computer. OBDH is the satellite platform computer in charge of processing platform and payload information and formatting/generating data to be transmitted to Ground Stations. The payload is composed of two scientific instruments (note the dashed rectangles). However, for the purpose of this case study, the main instrument is the one in which PDC exists, because SWPDC is embedded into PDC. The main goal of PDC is to obtain scientific data from EPPs and to transmit them to the OBDH. EPPs are front-end processors in charge of fast data processing of X-ray camera signals.

Essentially, this system employs a two-level primary/secondary communication model. In the first level, OBDH is the primary, PDC and IONEX are the secondaries. In the second level, PDC is the primary and EPPs (EPP H1 and EPP H2) are the secondaries. Communication protocols were specified to make the interface among the several computing units within the architecture.

The main functions of the SWPDC software product are: (i) interaction with EPPs in order to collect Scientific, Diagnosis and Test data; (ii) data formatting; (iii) memory management to store data temporarily before transmission to the OBDH; (iv) implementation of flow control mechanisms; (v) Housekeeping data generation; (vi) implementation of complex fault tolerance mechanisms; and (vii) loading of new programs on the fly (Santiago et al, 2007).

Fig. 2 shows QSEE's software development lifecycle processes (rectangles), formal technical reviews (circles), and main deliverables (artifacts). It is worth mentioning that the Independent Verification and Validation process was conducted by an independent team at INPE and started since the beginning of the software development lifecycle. Formal technical reviews were the main interaction points between customer and supplier. Each formal technical review had associated a set

of deliverables which were assessed by reviewers before the meeetings, and discussed during formal technical reviews meetings in order to improve their overall quality.



Caption:

SRR = System Requirements Review
PDR = Preliminary Design Review
DDR = Detailed Design Review
CDR = Critical Design Review
QR = Qualification Review
AR = Acceptance Review
RB = Requirements Baseline
POCP = PDC-OBDH Communication Protocol Specification
PECP = PDC-EPPs Communication Protocol Specification
SwDevPlan = Software Development Plan
IVVPlan = Independent Verification and Validation Plan
SRS = Software Requirements Specification
SwDesign = Software Design Document
SwTSP = Software Test Specification and Plan
SwTR = Software Test Report
SWPDCCode = SWPDC Source Code
UsrMan = SWPDC User Manual
SwAccPlan = Software Acceptance Test Plan
SwAccSpec = Software Acceptance Test Specification
SwAccReport = Software Acceptance Test Report

Fig. 2 QSEE project: software development lifecycle processes, reviews, and deliverables. Source: adapted from Santiago et al (2007)

The most important deliverables evaluated within each formal technical review are below each review's circle in Fig. 2. For example, within the Preliminary Design Review (PDR), PDC-OBDH Communication Protocol Specification (POCP), PDC-EPPs Communication Protocol Specification (PECP), Software Development Plan (SwDevPlan), Software Requirements Specification (SRS), and Independent Verification and Validation Plan (IVVPlan) were the main input and output

deliverables within PDR. Suppliers provided the deliverables marked with an asterisk (*), e.g. Software Requirements Specification, and the customer was in charge of the others with no asterisk, e.g. PDC-OBDH Communication Protocol Specification. Furthermore, deliverables in boldface and underlined mean that their output version within the review is considered their final version. Hence, Requirements Baseline (RB) was developed by the customer, it was input and output of the System Requirements Review (SRR), and RB's output version within SRR was frozen. The IVVPlan was also developed by the customer, and it was assessed within SRR, PDR, and Critical Design Review (CDR). IVVPlan's output version within CDR was frozen. SRS was responsibility of the suppliers and it was evaluated within PDR and Detailed Design Review (DDR). SRS's output version within DDR was frozen.

In the next two sections, we will describe the SOLIMVA methodology and its supporting tool and, whenever necessary, SWPDC case study will be used to demonstrate the accomplishment of the activities. In order to apply the SOLIMVA methodology, we consulted four deliverables: Requirements Baseline, Software Requirements Specification, PDC-OBDH Communication Protocol Specification, and PDC-EPPs Communication Protocol Specification.

3 The SOLIMVA methodology

The SOLIMVA methodology is illustrated in the activity diagram of Fig. 3. The first activity is the definition of a Dictionary by a user/test designer. The Dictionary defines the application domain and it is considered as a quintuple $\langle N, R, S_{TM}.C, S_{TM}.F, S_{TM}.Y \rangle$, where:

- N is a set of Names defining mainly the names of states of the model;
- $R : R.I_E \to R.O_E$. R is a function from $R.I_E$ (input event set) to $R.O_E$ (output event set) that represents the Reactiveness of the system;
- S_{TM} is the *Semantic Translation Model*. S_{TM} is composed of two sets and a function. One set characterizes specific control behaviors, $S_{TM}.C$. The other set characterizes the occurrence of self transitions within the model, $S_{TM}.F$; and $S_{TM}.Y : Y.I_P \to Y.O_P$. $S_{TM}.Y$ is a function from $Y.I_P$ (input pattern set) to $Y.O_P$ (output pattern set) that is related to hierarchy (depth) in the Statecharts model.

The set N, the functions R and $S_{TM}.Y$ are defined by the user. The sets $S_{TM}.C$ and $S_{TM}.F$ are already defined within the tool that supports the methodology, although the user can change if needed. Users enter data via Graphical User Interfaces and using NL. User is not required any knowledge in formal methods and their notations to define the application domain. It is worth mentioning that the Reactiveness feature of the Dictionary comes into picture because reactive systems are the main targets of SOLIMVA.

Let us consider the SWPDC case study described in Section 2. The Name (N) set of the Dictionary will be composed mainly by relevant words or set of words that map to important entities of the application domain. These include the first-level primary computing unit (OBDH), the computer in which SWPDC will be embedded (PDC) and the operation modes of such computer, SWPDC itself, and so on. Hence, N can be composed of:

N={PDC, SWPDC, OBDH, Initiation Operation Mode, Safety Operation Mode, ...}.

The Reactiveness (R) function is basically a mapping between the commands (the domain of R, i.e. $R.I_E$) and responses (the codomain of R, i.e. $R.O_E$) of the PDC-OBDH and PDC-EPPs

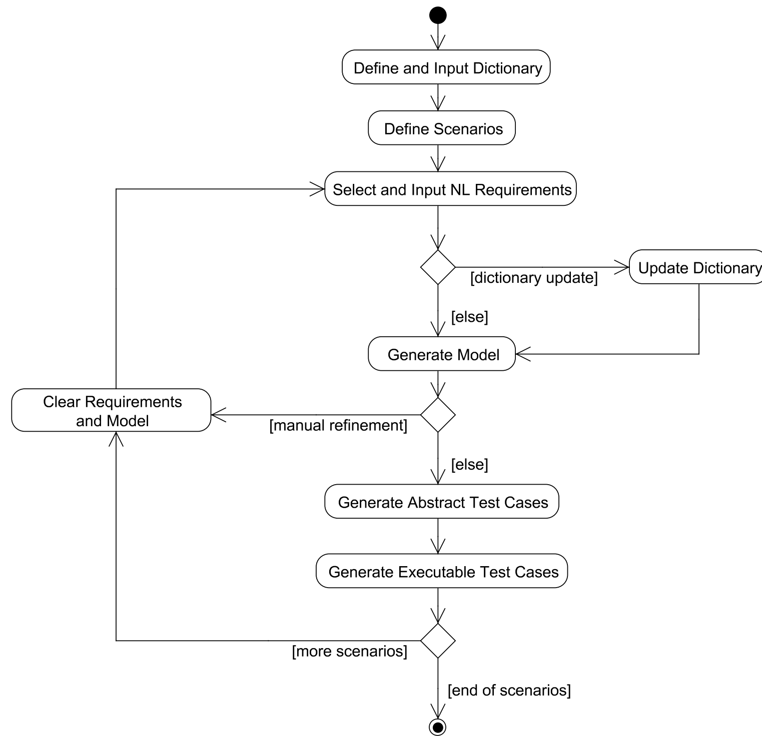


Fig. 3 The SOLIMVA methodology

Communication Protocol Specifications. For instance, VERIFY PDC's OPERATION MODE (VER-OP-MODE) is a command (an element of $R.I_E$) that the OBDH sends to PDC in order to know which is its current operation mode. The response that PDC sends back to the OBDH is precisely the INFORMATION REGARDING THE PDC's OPERATION MODE (INFO-OP-MODE), an element of $R.O_E$. The OBDH may CHANGE PDC's OPERATION MODE TO NOMINAL (CH-OP-MODE-NOMINAL) or CHANGE PDC's OPERATION MODE TO SAFETY (CH-OP-MODE-SAFETY). Assuming there is no problem during the transmission of the command to the PDC, in both cases the PDC responds with a positive acknowledgement, i.e. COMMAND CORRECTLY RECEIVED (CMD-REC). Hence, $R.I_E$, $R.O_E$, and R can be composed of:

$R.I_E$ = {VER-OP-MODE, CH-OP-MODE-NOMINAL, CH-OP-MODE-SAFETY, ...},
$R.O_E$ = {INFO-OP-MODE, CMD-REC, ...},
R = {(VER-OP-MODE, INFO-OP-MODE), (CH-OP-MODE-NOMINAL, CMD-REC), (CH-OP-MODE-SAFETY, CMD-REC), ...}.

The *Semantic Translation Model* of the Dictionary will be detailed in subsequent sections/subsections. After the definition of the Dictionary, scenarios are identified. Before continuing with our discussion, we need to define the meaning of a scenario in the context of this work.

Definition 1 *A **scenario** is defined as an interaction between a user and the Implementation Under Test (IUT). Associated to each scenario there is a set of requirements which characterize such an interaction.*

Fig. 4 details the *Define Scenarios* activity of the SOLIMVA methodology. The first task, which is optional, serves to obtain the basic elements that enable interaction with the IUT. In terms of embedded reactive systems, these basic elements can be protocol data units, commands, etc. that characterize the interface between two or more computing systems. Hence, a set of very **simple scenarios** are determined aiming at observing the correct implementation of these elements by the IUT.

We say that the first task is optional as the test designer might simply rely on test cases applied on previous phases of the software development lifecycle, e.g. unit testing, and consider that these basic elements are correctly implemented. In the SWPDC case study, a simple scenario is to switch the PDC on and send the VER-OP-MODE command to evaluate whether SWPDC has correctly implemented the reception and processing of this command.

```
   input  : NL requirements deliverables
   output: scenarios for model-based test case generation
 1 if identification of simple scenarios is needed then
 2 |  Identify simple scenarios based on the core elements that enable interaction with the IUT;
 3 end
 4 Identify factors and levels;
 5 Define strength = #factors − 1;
 6 Run the combinatorial designs algorithm, using the strength defined in the previous step;
 7 Identify normal scenarios. Consider the interpretation of at least (factors − 1) out of factors levels of
   each factor combination when identifying normal scenarios. Each factor combination will drive the
   identification of a scenario;
 8 foreach normal scenario do
 9 |  if unfolding is needed then
10 |  |  Identify new factors and levels;
11 |  |  Define a priority factor;
12 |  |  Define strength = #factors − 1;
13 |  |  Run the combinatorial designs algorithm, using the strength defined in the previous step;
14 |  |  Identify scenarios obtained by unfolding the normal scenario. At least (factors − 1) out of factors
   |  |  levels of each factor combination shall be accounted for when identifying unfolded scenarios. The
   |  |  number of unfolded scenarios will be the number of levels of the priority factor;
15 |  end
16 end
```

Fig. 4 The *Define Scenarios* activity of the SOLIMVA methodology in detail

In SOLIMVA, combinatorial designs are used to help to identify scenarios. Combinatorial designs are a set of techniques for test case generation which allow the selection of a small set of test cases even when the input domain, and the number of subdomains in its partition, is large and complex (Mathur, 2008). The basic idea is to define *factors* (input variables) and *levels* (values assignable to a factor) and to use a combinatorial designs algorithm to determine the set of levels, one for each factor, known as a *factor combination* or *run*.

Among the combinatorial designs techniques available, the SOLIMVA methodology adopted the *Mixed-Level Covering Array* which allows factors to assume levels from different sets. The algorithm

used is the *In-Parameter-Order* (IPO) (Lei and Tai, 1998), a procedure that can generate Mixed-Level Covering Arrays[1].

Let f_i, $1 \leq i \leq m$, be a set of factors, and l_{ij}, $1 \leq j \leq n$, be the set of levels for each factor f_i, where n may vary depending on i. Hence, a factor combination number X (*fc X*) of the generated Mixed-Level Covering Array could be *fc X*$= \{l_{11}, l_{23}, \ldots, l_{m1}\}$ which means the first level of factor 1, the third level of factor 2, and so on until the first level of factor m. Based on *fc X*, the test designer must search for requirements within the NL requirements deliverables that address each level of *fc X* and thus characterize a scenario. In other words, a factor combination derived by the combinatorial designs algorithm is interpreted by the test designer in order to define a scenario (the interaction between the user and the IUT). In this paper when we refer to for example "scenario X", where X is given by the tool that has implemented the combinatorial designs algorithm, we mean that this is the scenario defined by the test designer due to the interpretation of *fc X*. This perspective applies only to the **normal scenarios** and **unfolded scenarios** (further explanation of these scenarios in sequence) which are those generated with the aid of the combinatorial designs algorithm. In other words, there is no *fc X* to interpret to define simple scenarios because these are not derived with the help of the tool that has implemented the combinatorial designs algorithm.

As seen from lines 4 to 7 in Fig. 4, scenarios identified by the interpretation of the factor combinations due to the first use of combinatorial designs are called **normal scenarios**. Besides, note that $strength = \#factors - 1$ with the objective of achieving the maximum number of factor combinations without using an exhaustive (all to all) approach (#factors = total number of factors).

Notice that a scenario might have more that one test objective associated. However, these test objectives should not be too disparate. This recommendation might lead the user to neglect one level on interpreting the factor combination in order to identify a scenario. The explanation for this fact is that the characteristics of the factors can be considerably different so that if the test designer defines a scenario based on all levels of a factor combination, it is possible that such a scenario has several unrelated test objectives which is not a good approach, resulting in a bad strategy in terms of test objectives. For instance, in a web application, interpretation of a factor combination may generate a scenario in which it is necessary to verify whether a web service is correctly implemented, and some security requirements are satisfied, and information retrieval from a data base due to a specific type of request is consistent. There are many different and unrelated test objectives in this case. Thus, it is more interesting to disregard the interpretation of a level to decrease the amount of test objectives. In such situations, we use "-" to mean "do not consider any level of this factor for this particular scenario".

A simplified choice of factors and levels for the SWPDC case study is shown in Table 1. The explanation for the factors and levels follows:

1. Cmd. This factor relates to the commands defined in the PDC-OBDH Communication Protocol Specification. These commands were grouped into levels considering processing activities to acquire and transmit data (DtAcqTx), and handle hardware and software parameters (HwSwHnd);
2. OpMode. This factor relates to PDC's operation modes. In this simplified example, we considered only the Nominal Operation Mode (Nom);

[1] At present, SOLIMVA uses an open source tool, TConfig (University of Ottawa, 2008), which has implemented the IPO procedure. The IPO procedure was originally conceived for 2-way (pairwise) testing. The TConfig tool can generate from 2-way to 6-way Mixed-Level Covering Arrays. However, it is not clear if the tool has implemented the *In-Parameter-Order-General* (IPOG) (Lei et al, 2007) algorithm which deals with general t-way testing or else if another approach was adopted.

3. Services. This factor relates to the services supported by SWPDC. In this example, we took into account only the services related to acquisition, formatting, and transmission of Scientific Data (Sci), and generation, formatting, and transmission of Housekeeping Data (Hk).

Table 1 Simplified choice of factors and levels for the SWPDC case study

Factors	**Levels**		
Cmd	DtAcqTx	HwSwHnd	Inv
OpMode	Nom	Inv	
Services	Sci	Hk	Inv

Note that each factor has a level *Inv*, which stands for invalid value. The SOLIMVA methodology strongly recommends that each factor defines such a level to address Robustness testing which is particularly useful for embedded critical software where, for instance, it is possible to observe the behavior of the IUT under non specified test input data. Running the combinatorial designs algorithm with $strength = \#factors - 1 = 2$, nine factor combinations are generated. Table 2 shows the normal scenarios based on the interpretation of each factor combination.

The first remark about the generated scenarios is that when a level is not present in a factor combination, this does not necessarily imply that it will not be somehow related to the scenario derived from the interpretation of such factor combination. It depends on the kind of factor. For instance, in Table 2, normal scenario 1 has level DtAcqTx due to the command factor (Cmd). It does not mean that the selection of NL requirements that characterize normal scenario 1 will be such that no requirement shall be related to commands to handle hardware and software parameters (HwSwHnd). Indeed, it is very likely that HwSwHnd commands should be sent to PDC in order to drive the SWPDC to the appropriate state so that the test objective of normal scenario 1 can be achieved. For example, in order to acquire, format, and transmit Scientific Data in the Nominal Operation Mode, first EPP H1 and EPP H2 must both be turned on. But, the commands to switch them on are HwSwHnd commands. Hence, they need to be sent to PDC prior to data acquisition. However, for normal scenario 1, the main contribution of the command factor is related to data acquisition and transmission (DtAcqTx).

Notice that in normal scenarios 2, 3 and 4, one level was not accounted for ("-") when interpreting the factor combinations. This level was precisely *Inv* which was addressed in other scenarios (5, 6, 7, 9) whose main goals were related to Robustness testing. This stresses the important recommendation that the methodology provides to incorporate Robustness testing covering several different situations. Scenarios where Robustness is the main test objective can be mapped to "unhappy cases" in use case modeling.

The *Inv* level may be translated into many different test input data. This is a characteristic of combinatorial designs testing where each factor combination may drive one or more test cases where each test case consists of test input data and the expected result (Mathur, 2008). For instance, in normal scenario 5, the test designer may select several different invalid commands by looking at the PDC-OBDH Communication Protocol Specification and choosing "commands" that are not specified to be sent to PDC. To do that, other traditional black box testing techniques like boundary-value analysis might be applied to choose such invalid values. Hence, normal scenario 5 may have as many as needed invalid commands the test designer wishes, addressing the robustness of SWPDC. Another

Table 2 Normal scenarios due to the factors and levels of Table 1

Factor Combination	Scenario	Interpretation
{DtAcqTx, Nom, Sci}	1	Acquire, format, and transmit Scientific Data in the Nominal Operation Mode
{DtAcqTx, -, Hk}	2	Generate, format, and transmit Housekeeping Data in the Nominal Operation Mode
{HwSwHnd, Nom, -}	3	Verify the correct implementation of commands related to hardware parameters manipulation in the Nominal Operation Mode
{HwSwHnd, -, Sci}	4	Verify the correct implementation of commands related to hardware parameters manipulation during acquisition, formatting, and transmition of Scientific Data
{Inv, Nom, Hk}	5	Verify the behavior of SWPDC when receiving commands with inconsistent values during transmission of Housekeeping Data in the Nominal Operation Mode (Robustness testing)
{Inv, Inv, Inv}	6	Verify the behavior of SWPDC when receiving commands with inconsistent values, when trying to change PDC's operation mode to an unspecified operation mode, and when asking SWPDC to provide services not defined in the Software Requirements Specification (Robustness testing)
{Inv, Nom, Sci}	7	Verify the behavior of SWPDC when receiving commands with inconsistent values during acquisition, formatting, and transmission of Scientific Data in the Nominal Operation Mode (Robustness testing)
{HwSwHnd, Nom, Hk}	8	Verify the correct implementation of commands related to software parameters manipulation, and generate, format, and transmit Housekeeping Data in the Nominal Operation Mode
{DtAcqTx, Nom, Inv}	9	Verify the behavior of SWPDC when asking SWPDC to provide services not defined in the Software Requirements Specification during data acquisition, generation, and transmission (Scientific or Housekeeping Data) in the Nominal Operation Mode (Robustness testing)

approach if the Cmd factor is *Inv* within a factor combination is to address situations where a command is not entirely received by PDC due to problems in the physical transmission medium.

The fact that the user can neglect one level on interpreting the factor combination to derive a scenario does not mean that the entire test suite, considering all test cases derived according to all Statecharts models, would be incomplete. Looking at Table 2, we observe that the set of derived scenarios cover all aspects of factors/levels of Table 1. In other words, within the scenarios defined in Table 2, it is possible to acquire, format, and transmit Scientific Data in the Nominal Operation Mode, to generate, format, and transmit Housekeeping Data in the Nominal Operation Mode, to verify the correct implementation of HwSwHnd commands, related to hardware and software parameters, and of DtAcqTx commands too. What matters is the expertise of the test designer in the application domain in order to interpret the factor combinations and to define scenarios that will generate, at the end, a test suite that is sufficiently complete.

However, it is possible that some normal scenarios have to be unfolded so that more factor combinations should be generated. The explanation for the need of such unfolding process lies in the fact that some levels may implicitly have specific values of variables (e.g. initial and final values of memory addresses) so that it is necessary to deal with situations which address the combination of such values. Not all normal scenarios need to have this demand. For those that need, new factors

and levels are defined, as well as a priority factor. The total number of **unfolded scenarios** due to such process must be equal to the number of levels of the priority factor.

In case a normal scenario is unfolded, it is not necessary to accomplish that more than once because multiple unfoldings will make the methodology complex without substantial benefit in practical terms. Normal scenario 8 in Table 2 is the only one that needs to be unfolded into other scenarios. This is because the software parameters that are updated via commands have a default value, but also minimum and maximum values. Hence, it is interesting to replace normal scenario 8 with other more specific scenarios addressing several situations regarding such values. Table 3 shows a simplified set of factors and levels for unfolding normal scenario 8.

Table 3 Unfolding normal scenario 8: simplified choice of factors and levels

Factors	**Levels**			
HkTime	Min	Def	Max	Inv
IniPtr	Min	InRng	Max	Inv
SmpTime	Min	Def	Max	Inv

In Table 3, the priority factor is the parameter that defines the period (time interval) in which Housekeeping Data are generated (HkTime). In the PDC-OBDH Communication Protocol Specification, not only a default value (Def) is specified but also minimum (Min) and maximun (Max) values of this parameter. The same levels of HkTime applies to the sampling time of the analog input channels (SmpTime). The initial pointer (memory address) in which it is possible to load new executable code on the fly (IniPtr) has also minimum (Min) and maximum (Max) values, but it also allows initial addresses in range (InRng), i.e. between the minimum and maximum values.

Since the priority factor has 4 levels then four unfolded scenarios will be defined and will replace normal scenario 8. For instance, unfolded scenario 8.2, i.e. the second scenario unfolded from normal scenario 8, suggests the test designer to add requirements related to commands so that the following situations are covered:

- HkTime = Def, IniPtr = Min, SmpTime = Def;
- HkTime = Def, IniPtr = InRng, SmpTime = Min;
- HkTime = Def, IniPtr = Max, SmpTime = Inv;
- HkTime = Def, IniPtr = Inv, SmpTime = Max.

Note that the time to generate Housekeeping Data remains fixed in the default value within unfolded scenario 8.2. The other parameters must be udpated via commands with different values. Besides, Robustness testing is still in order due to the invalid values. In these cases, SWPDC can receive but it must not process any of the invalid commands.

After the previous steps, the user must select and input a set of NL requirements which together characterize a single scenario (simple, normal, unfolded). Then, the user must search these requirements in documents such as software requirements specifications. For example, in the SWPDC case study, each interaction with the IUT requires that the PDC is energized and, after that, the activities of initializing the computing system are performed by SWPDC. Thus the following requirements, defined in the SWPDC's Software Requirements Specification, relate to the beginning of each scenario (SRSxxx is the requirement identification):

SRS001 - The PDC shall be powered on by the Power Conditioning Unit.
SRS002 - The PDC shall be in the Initiation Operation Mode after being powered on. The SWPDC shall then accomplish a POST. If PDC presents any irrecoverable problem, this computer shall remain in the Initiation Operation Mode and such a problem shall not be propagated to the OBDH.
SRS003 - If PDC does not present any irrecoverable problem, after the initiation process, the PDC shall automatically enter into the Safety Operation Mode.

Section 5 addresses the SWPDC case study in more detail, showing sets of chosen NL requirements that characterize scenarios.

The Dictionary does not necessarily have to be defined completely at once. The user can start defining and inputting part of the Dictionary at the beggining and, after choosing the NL requirements that characterize each scenario, the Dictionary can be updated according to new important words. This is the reason behind the optional activity *Update Dictionary*. Hence, the creation of the Dictionary is incremental and dependent on the selected set of NL requirements. This approach prevents the user to completely define the Dictionary at an early stage when applying SOLIMVA.

After that, the generation of the Statecharts model follows. This activity will be discussed in detail in Section 4. After generating the model, the test designer may decide to manually refine it. Hence, he/she can accomplish this refinement in which the requirements of the scenario and the respective created model must be cleared. The user can make such a refinement because he/she realized that the generated model can be improved and/or there is some kind of incoherence in the model probably due to mistakes when inserting the NL requirements (e.g. wrong sequence of requirements). Let us illustrate the relevance of the manual refinement with requirements SRS001, SRS002, and SRS003 presented above. If the test designer mistakenly switch the order of NL requirements SRS002 and SRS003, the resulting Statecharts model due to this wrong input sequence of requirements will be naturally incorrect. Fig. 5 shows a piece of the correct Statecharts model while Fig. 6 shows a piece of the incorrect Statecharts model due to the wrong order of NL requirements.

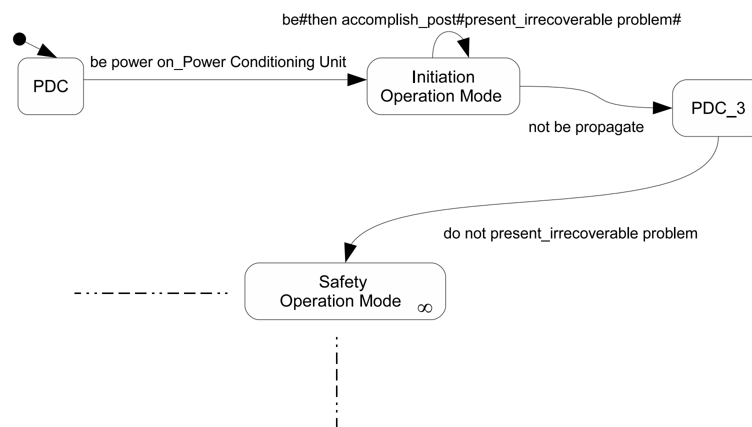


Fig. 5 Piece of the entire Statecharts model related to a scenario: correct model

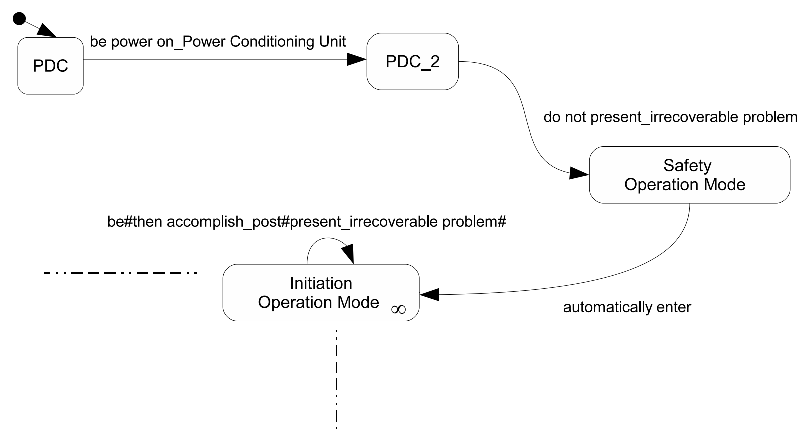


Fig. 6 Piece of the entire Statecharts model related to a scenario: incorrect model

After these steps, **Abstract Test Cases** are generated by using the GTSC environment (Santiago et al, 2008b). GTSC 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. At present, GTSC has implemented Distinguishing Sequence (DS), Unique Input/Output (UIO) (Sidhu and Leung, 1989) and H-switch cover (Souza, 2010) test criteria for FSM models, and four test criteria from the *Statechart Coverage Criteria Family* (SCCF) (Souza, 2000), all-transitions, all-simple-paths, all-paths-k-C0-configuration, and all-paths-k-configurations, targeting Statecharts models. In other words, test criteria define the rules that drive test case generation in GTSC.

In order to use GTSC, a user translates the Statecharts behavioral model into the *PerformCharts Markup Language* (PcML) (Santiago et al, 2006). In case of SOLIMVA, the idea is to automatically translate the generated model into PcML. Based on a PcML document, a flat FSM is generated by GTSC. A flat FSM is a model where all hierarchical and orthogonal features of a Statecharts model were removed. PerformCharts tool (Vijaykumar et al, 2006), one of the components of the GTSC environment, is responsible for that. This flat FSM is indeed the basis for test case generation.

The generated Statecharts model is an abstract representation of the behavior of the IUT according to a specific scenario. Hence, the test cases derived from this model are a kind of functional tests on the same level of abstraction as the model. As will be shown in Section 5, the generated "test input data" and "expected results" of the test cases based on the Statecharts model are usually pieces of NL sentences (particularly in the SWPDC case study, some commands/responses of Communication Protocol Specifications may also be present). Hence, the test cases generated by GTSC are Abstract Test Cases and thus they cannot be directly executed against the IUT due to the fact they are on the incorrect level of abstraction. Therefore, the test designer shall accomplish the translation from Abstract Test Cases into **Executable Test Cases** to enable the effective execution of test cases.

Having created the test cases (Executable Test Cases) for a single scenario, the test designer starts again selecting and inserting the NL requirements for the next scenario. But before doing this, he/she must clear the requirements and related model of the current scenario. This process must be repeated until there is no more scenario.

4 Model generation

The *Generate Model* activity in Fig. 3 is composed of two sub-activities. When the user selects the options related to these sub-activities in SOLIMVA's Graphical User Interfaces, algorithms are executed to meet the goals. These sub-activities are described below.

4.1 Generation of tuples

The first task refers to the generation of *Behavior-Subject-Action-Object* (BSAO) 4-tuples. The BSAO tuples are an extension of the concept of SAO triads used in the *Java Requirement Analyzer* (J-RAn) tool (Fantechi and Spinicci, 2005).

In SOLIMVA, the first extension is the inclusion of Behavior features in the SAO triad transforming into a BSAO 4-tuple. The reason behind this lies in the fact that words like *if* determine a particular behavior in the created model. For instance, finding an if-then-else situation in one or in several NL requirements (e.g. in a requirement: "If an echo is received ..."; in the same or in the next requirement: "If an echo is not received ...") may imply that the behavioral model will have a state with two outgoing transitions each one representing the possible outcome of the if-then-else situation. Hence, if we identify such a situation within the NL requirements that characterize a scenario, we follow the same approach when building a Control Flow Graph from the source code and dealing with a control structure *if*.

The second modification is related to Object identification. J-RAn presented a large number of extractions that were not detected (Fantechi and Spinicci, 2005), and one explanation for this fact is because J-RAn used a single link type ("O") of the *Link Grammar Parser* (Sleator and Temperley, 1993) to identify Objects. However, it is possible that there is no explicit Object generated by *Link Grammar* depending on the NL requirement. Consider the following requirement:

Users' data shall be updated on the server every 12 hours.

When using *Link Grammar*, there is no Object because none of the link types with respect to Object ("O", "OT", ...) appears in the parser output. We tried to overcome such situations in the tool that supports SOLIMVA. We developed and implemented an algorithm to automatically identify the BSAO tuples. The algorithm makes use of the *Stanford Part Of Speech (POS) Tagger* (Toutanova et al, 2003) in order to identify the lexical categories (i.e. the parts of speech[2]) of each sentence of the NL requirements. Actually, after the user has selected the NL requirements that characterize a scenario, the SOLIMVA tool combines all such requirements into a file. This file is input to the *Stanford POS Tagger* which assigns the POS tag of each word[3].

Fig. 7 shows the algorithm to automatically generate the BSAO tuples based on the NL requirements. Besides the Dictionary (*dic*), the algorithm takes as input all the words (*allw* set) in the file that contains the set of NL requirements. In case of verbs and nouns, the algorithm obtains the lemma[4] and adds it in the *allw* set instead of adding the word itself. For simplification, from now onwards when writing "word", we may refer to the word itself or to its lemma.

[2] Lexical category or part of speech is a linguistic category of words (or more precisely lexical items), which is generally defined by the syntactic or morphological behavior of the lexical item in question. Common linguistic categories include noun and verb, among others.

[3] The *Stanford POS Tagger* adopts the *Penn Treebank POS tagset* (Marcus et al, 1993).

[4] In linguistics, one definition of lemma is: the canonical form, dictionary form, or citation form of a set of words (headword). For instance, in English, "run", "runs", and "ran" are forms of the same lexeme, with "run" as the lemma.

With respect to the notation in Fig. 7, the $allw$ set is in fact a set of ordered triples where each ordered triple is composed of: a word (or lemma) (upper index lw), a POS tag (upper index tg) obtained from the POS tagging algorithm implemented in the SOLIMVA tool, and a counter for the exact identification of the word (upper index id). Hence, $allw$ is a set with the following elements:

$$allw = \{(w_1^{lw}, w_1^{tg}, w_1^{id}), (w_2^{lw}, w_2^{tg}, w_2^{id}), \ldots, (w_h^{lw}, w_h^{tg}, w_h^{id})\}.$$

Thus, w_1^{lw} means the word of the first ordered triple, w_4^{tg} means the POS tag of the fourth ordered triple, and w_7^{id} represents the identification of the seventh ordered triple of the set. In the tuples (tup) set, b, s, a, o represent Behavior, Subject, Action, and Object, respectively. The verbs for Word Sense Disambiguation (WSD) set ($vrbwsd$) will be discussed in Subsection 4.2.2. Some additional sets were predefined in order to help the process of BSAO tuples derivation. They are:

1. $usefulWords$. This set contains POS tags that aid in the process of creating a key for the words within the requirements. The key is defined as "counter of Useful Words ($cntUW$) - word". It also defines the lexical categories of the running/useful words for the WSD algorithm (Subsection 4.2.2). The chosen lexical categories were: common nouns (singular and plural), verbs, adjectives, and adverbs;
2. $candSubObj$. This set contains POS tags that define the candidate words to be Subjects and Objects. The selected lexical categories were: common nouns (singular and plural), proper nouns (singular and plural), adjectives, cardinal numbers, and coordinating conjunctions;
3. $confirmSub$. This set contains POS tags that confirm that a previous word identified as a Subject is indeed a Subject. The selected lexical categories were: modal verbs and verbs. This set is important because after a Subject usually there is a verb. If no verb is found, it is likely that a word previously identified as a Subject is not in fact a Subject;
4. $confirmAct$. Set that contains POS tags that characterize an Action. The selected lexical categories were: verbs, adverbs, and coordinating conjunctions;
5. $endTuple$. Set that contains POS tags which aid in the decision whether a BSAO tuple must be created or not. The chosen lexical categories were: common nouns (singular and plural), proper nouns (singular and plural), verbs, adjectives, adverbs, modal verbs, and cardinal numbers.

One of the first tasks accomplished within the algorithm is the probable identification of a Behavior (b) feature. Thus, the algorithm first verifies if a word is a *preposition or subordinating conjunction* (POS tag "IN") and also if such a word is in $S_{TM}.C$ (lines 7 to 9 in Fig. 7). If these conditions are matched, then the b element of the tuple is assigned to such a word. If not, b is empty. Note that the $S_{TM}.C$ set has words like "if", and, at first, the user does not need to alter it. This set is independent of the application domain and it was derived aiming to add in the resulting Statecharts model behaviors due to the semantics associated to some requirements.

After the determination of b, the Subject (s), Action (a), and Object (o) elements of the tuple are identified (lines 10 to 48). Essentially, the algorithm verifies whether the POS tags of words are equal to predefined POS tags that characterize a Subject (according to the sets $candSubObj$ and $confirmSub$), an Action (according to the set $confirmAct$), or an Object (based on set $candSubObj$). If they match, the corresponding s, a, and o elements of the tuple are fulfilled.

However, a BSAO tuple is created **if and only if** a Subject and an Action and an Object were identified, and some other conditions were satisfied (lines 50 to 53). If these conditions are not satisfied, no BSAO tuple is created. This is to avoid situations which might occur in NL sentences, and might produce ill-formed tuples. For instance, a piece of a sentence might derive an s, an a but not an o

```
input  : dictionary dic = {N, R, S_TM.C, S_TM.F, S_TM.Y}
input  : allWords allw = {w_f^g | g = lw, tg, id}, f = 1..h
output: tuples tup = {t_i^j | j = b, s, a, o}, i = 1..k
output: verbsWSD vrbwsd = {v_p^q | q = ivr, inf}, p = 1..r
1  initializeAuxiliaryVariables();
2  for f ← 1 to h do
3      while w_f^tg ≠ "." do
4          if w_f^tg ∈ usefulWords then
5              cntUW ← cntUW + 1;
6          end
7          if w_f^tg = "IN" ∧ w_f^lw ∈ S_TM.C then
8              behavior ← behavior + w_f^lw;
9          end
10         if w_f^tg ∈ candSubObj ∧ ¬subCreated then
11             if iLastSub = 1 ∨ iLastSub = f − 1 then
12                 subject ← subject + w_f^lw;
13                 iLastSub ← f;
14             else
15                 if iLastSub ≠ f − 1 then
16                     subject ← empty;
17                     subject ← w_f^lw;
18                     iLastSub ← f;
19                 end
20             end
21         else
22             if w_f^tg ∈ confirmSub ∧ subject ≠ empty then
23                 subCreated ← true;
24                 if w_f^tg ∈ confirmAct then
25                     if iLastAct = 1 ∨ iLastAct = f − 1 then
26                         action ← action + "cntUW" + " − " + w_f^lw;
27                         iLastAct ← f;
28                         actCreated ← true;
29                     end
30                 end
31             else
32                 if w_f^tg ∈ confirmAct ∧ subCreated then
33                     if iLastAct = 1 ∨ iLastAct = f − 1 then
34                         action ← action + "cntUW" + " − " + w_f^lw;
35                         iLastAct ← f;
36                         actCreated ← true;
37                     end
38                 else
39                     if w_f^tg ∈ candSubObj ∧ actCreated then
40                         if iLastObj = 1 ∨ iLastObj = f − 1 then
41                             object ← object + w_f^lw;
42                             iLastObj ← f;
43                             objCreated ← true;
44                         end
45                     end
46                 end
47             end
48         end
49         f ← f + 1;
50         if subCreated ∧ actCreated ∧ objCreated ∧ w_f^tg = "." ∨ w_f^tg = "," ∨ w_f^tg ∉ endTuple
           then
51             (t_i^b, t_i^s, t_i^a, t_i^o) ← (behavior, subject, action, object);
52             initializeAuxiliaryVariables();
53         end
54     end
55     initializeAuxiliaryVariables();
56 end
57 vrbwsd ← generateVerbsWSD(tup, dic);
```

Fig. 7 Main algorithm for generating BSAO tuples

because the sentence ended. The algorithm picks the next ordered triple of the *allw* set (line 49) in order to continue the assessment of words within a sentence (a sentence is delimited by a period). However, the algorithm also checks whether not only if s, a and o were created but also if the POS tag of the next ordered triple of the set is a period in order to decide about the creation of a BSAO tuple.

Let us illustrate how a BSAO tuple is automatically generated. Consider the following requirement:

> If the main software system does not start operating the air conditioning system at midnight, maintenance personnel should be called.

Table 4 shows how a BSAO tuple is generated considering each word of this requirement. The POS tag and consequently the lexical category of each word is provided by the *Stanford POS Tagger*. The Behavior (B), Subject (S), Action (A), and Object (O) columns contain the value of each one of these tuple's elements after applying algorithm shown in Fig. 7.

The conditions that determine behavior (line 7) were satisfied and hence the Behavior element of the tuple was fulfilled accordingly. Furthermore, note that the algorithm can handle compoud names for Subjects. Line 12 of the algorithm shows that the auxiliary variable *subject* is updated so that the Subject element of the tuple (t_i^s, line 51) can be filled with the correct composition of words. Conditions must be satisfied for a word to be part of the Subject: POS tag of such a word must be in *candSubObj* set; and, Subject must not have been yet created (variable *subCreated*; line 10). There is also a mechanism to avoid the inclusion of words whose POS tags are in the *candSubObj* set but do not really make part of the Subject (lines 11 to 13). These are the reasons why "main", which is an adjective, is also correctly included as part of the Subject element of the tuple. Similar remarks are valid for compound names for Actions (lines 24 to 37, and line 51), and compound names for Objects (lines 39 to 45, and line 51).

The content of Action was derived with the keys previously mentioned. In other words, "4-do" is the key due to the verb "does". The number (4) is the current "counter of Useful Words (*cntUW*)" which uniquely identifies this word within the set of NL requirements. In this case, "do" is the lemma of "does". Notice that the same explanation applies to "6-start" due to the verb "start", and "7-operate" due to "operating".

The NL requirement presented in Table 4 might generate another BSAO tuple with Behavior empty, "maintenance personnel" as Subject, and "14-be 15-call" as Action. However, the sentence ends and no Object has been detected. Therefore, a second BSAO tuple is not created because the Object element is missing.

4.2 Translation from BSAO tuples into behavioral model

The Dictionary and the BSAO tuples form the basis for model generation. Fig. 8 shows the main algorithm supporting this second sub-activity. The remarks about notation made in Subsection 4.1 apply to Fig. 8. One additional remark is about the notation related to functions. We consider a function F a set of ordered pairs (x, y), where x is an element of the domain of F, and y is an element of the codomain of F. With respect to the Reactiveness (R) function, the notation $R_n(rie)$ means this is the element of the domain of R ($rie \in R.I_E$) of the nth ordered pair of R. Similar observations apply to the codomain of R ($R.O_E$), and also to the $S_{TM}.Y$ function.

The generated model (*mod*) is a set of ordered quadruples where each ordered quadruple represents a transition in the model. Hence, each ordered quadruple is composed of: a source state (upper index

Table 4 Generation of a BSAO tuple

Word	Lexical Category	Tag	B	S	A	O
If	preposition or subordinating conjunction	IN	if			
the	determiner	DT	if			
main	adjective	JJ	if	main		
software	common noun, singular	NN	if	main software		
system	common noun, singular	NN	if	main software system		
does	verb, present tense, 3rd person singular	VBZ	if	main software system	4-do	
not	adverb	RB	if	main software system	4-do 5-not	
start	verb, base form	VB	if	main software system	4-do 5-not 6-start	
operating	verb, gerund or present participle	VBG	if	main software system	4-do 5-not 6-start 7-operate	
the	determiner	DT	if	main software system	4-do 5-not 6-start 7-operate	
air	common noun, singular	NN	if	main software system	4-do 5-not 6-start 7-operate	air
conditioning	common noun, singular	NN	if	main software system	4-do 5-not 6-start 7-operate	air conditioning
system	common noun, singular	NN	if	main software system	4-do 5-not 6-start 7-operate	air conditioning system
at	preposition or subordinating conjunction	IN				
midnight	common noun, singular	NN				
maintenance	common noun, singular	NN				
personnel	common noun, plural	NNS				
should	modal verb	MD				
be	verb, base form	VB				
called	verb, past participle	VBN				

src), an input event (upper index *iev*), an output event (upper index *oev*), and a destination state (upper index *des*). The initial idea is to denote the states of the model with the Subject of the BSAO tuple. This is clearly shown in lines 6, 14 and 31 in Fig. 8. However, the *checkSubject* algorithm (lines 6 and 14) checks whether there is already a source state in the model with the same name of the current BSAO Subject. The *checkSubject* algorithm returns a name for the state just adding an underscore followed by an incrementing number after the Subject, if there is already a same state name in the model; otherwise, it will return the same BSAO Subject to be assigned as the name of the source state.

The Reactiveness (R) function of the Dictionary plays an important role in defining input and output events within a transition. As shown from lines 22 to 28, if the Object of the BSAO 4-tuple exists in the input event set ($R.I_E$), then the input event (*iev*) will be assigned to the matched element of the domain of R ($R.I_E$), and the output event (*oev*) will have the value of the corresponding element of the codomain of R ($R.O_E$). However, if the tuple's Object does not match any element of $R.I_E$, the input event becomes a combination of Action_Object of the BSAO tuple, and the output event null. Another remark is that the if-then-else situation in NL sentences is addressed from lines 3 to 21. Hence, more than one transition may be leaving the same source state in the resulting model.

Let us consider the following two requirements from the SWPDC case study:

> SRS001 - The PDC shall be powered on by the Power Conditioning Unit.
> SRS002 - The PDC shall be in the Initiation Operation Mode after being powered on. The SWPDC shall then accomplish a POST. If PDC presents any irrecoverable problem, this computer shall remain in the Initiation Operation Mode and such a problem shall not be propagated to the OBDH.

These requirements produce 6 BSAO tuples as shown in Table 5. Piece of the resulting model set based on these tuples is shown in Table 6.

At this point, the number of transitions in the created model is equal to the number of BSAO tuples. Note that the Subjects of the BSAO tuples define the name of the source states in Table 6. However, the *checkSubject* algorithm creates different names for source states when finding "PDC" repeatedly by adding "_" followed by an incrementing counter. None of the Objects of BSAO tuples are in $R.I_E$. Hence, each input event is a concatenation of Action_Object of the BSAO tuple, and the output event is null. Besides, all destination states are also null.

Some ordered quadruples of the model may be removed. As shown from lines 36 to 38 of the main algorithm (Fig. 8), there are three types of automated refinements which are applied to the original model so that the final model may be enhanced with respect to the original one.

4.2.1 Refinement based on domain information

The first automated refinement is to eliminate unnecessary states and transitions of the model, to rename certain states of the model, and to fulfill the destination states of the transitions. We call it a *refinement based on domain information* ($refineModelDomainInfo$). It removes an ordered quadruple (a transition) from the original model if the source state does not exist in N, and the Object part of the input event does not exist in N, and the entire input event does not exist either in $R.I_E$. This is done because not all NL sentences contain relevant information to justify the creation of a transition in the context of model-based test case generation.

If an ordered quadruple is not to be removed, the name of its source state may be changed if the Object part of the input event exists in N and the entire input event does not exist in $R.I_E$. This

input : dictionary $dic = \{N, R, S_{TM}.C, S_{TM}.F, S_{TM}.Y\}$
input : tuples $tup = \{t_i^j \mid j = b, s, a, o\}, i = 1..k$
input : verbsWSD $vrbwsd = \{v_p^q \mid q = ivr, inf\}, p = 1..r$
output: model $mod = \{m_x^y \mid y = src, iev, oev, des\}, x = 1..z$

1 $cond \leftarrow false$;
2 **for** $i \leftarrow 1$ **to** k **do**
3   **if** $t_i^b \in S_{TM}.C \wedge \neg cond$ **then**
4     $existBeh \leftarrow t_i^b$;
5     $existSub \leftarrow t_i^s$;
6     $tmod^{src} \leftarrow$ `checkSubject(`t_i^s`)`;
7     $lastState \leftarrow tmod^{src}$;
8     $cond \leftarrow true$;
9   **else**
10   **if** $t_i^b = existBeh \wedge t_i^s = existSub$ **then**
11       $tmod^{src} \leftarrow lastState$;
12       $cond \leftarrow false$;
13   **else**
14       $tmod^{src} \leftarrow$ `checkSubject(`t_i^s`)`;
15   **if** $t_i^b \in S_{TM}.C$ **then**
16         $existBeh \leftarrow t_i^b$;
17         $existSub \leftarrow t_i^s$;
18         $lastState \leftarrow tmod^{src}$;
19   **end**
20   **end**
21   **end**
22   **if** $t_i^o \in R.I_E$ **then**
    // $rie \in R.I_E$.
23     $tmod^{iev} \leftarrow R_n(rie)$;
    // $roe \in R.O_E$.
24     $tmod^{oev} \leftarrow R_n(roe)$;
25   **else**
26     $tmod^{iev} \leftarrow t_i^a +$ "-" $+ t_i^o$;
27     $tmod^{oev} \leftarrow null$;
28   **end**
29   $tmod^{des} \leftarrow null$;
  // add element to model mod.
30   $x \leftarrow i$;
31   $m_x^{src} \leftarrow tmod^{src}$;
32   $m_x^{iev} \leftarrow tmod^{iev}$;
33   $m_x^{oev} \leftarrow tmod^{oev}$;
34   $m_x^{des} \leftarrow tmod^{des}$;
35 **end**
// first automated refinement.
36 $mod \leftarrow$ `refineModelDomainInfo`(mod, dic);
// second automated refinement.
37 $mod \leftarrow$ `refineModelWordSense`$(mod, dic, vrbwsd)$;
// third automated refinement.
38 $mod \leftarrow$ `refineModelHierarchy`(mod, dic);

Fig. 8 Main algorithm for the translation of BSAO tuples into models

Table 5 BSAO tuples derived from requirements SRS001 and SRS002

B	S	A	O
-	PDC	1-be 2-power on	Power Conditioning Unit
-	PDC	3-be	Initiation Operation Mode
-	SWPDC	6-then 7-accomplish	post
if	PDC	9-present	irrecoverable problem
-	computer	13-remain	Initiation Operation Mode
-	problem	15-not 16-be 17-propagate	OBDH

Table 6 Piece of the model set derived from the BSAO tuples in Table 5. Caption: #Tr = Transition number

#Tr	Source State	Input Event	Output Event	Destination State
1	PDC	1-be 2-power on_Power Conditioning Unit	null	null
2	PDC_2	3-be_Initiation Operation Mode	null	null
3	SWPDC	6-then 7-accomplish_post	null	null
4	PDC_3	9-present_irrecoverable problem	null	null
5	computer	13-remain_Initiation Operation Mode	null	null
6	problem	15-not 16-be 17-propagate_OBDH	null	null

is explained due to the fact that Subjects, which in turn first generated the name of states in the model, in NL requirements are usually a few names like system, the name of a computer or a software product. This implies that the name of the states would be basically limited to those names added by a counter (recall the *check_Subject* algorithm) such as system, system_1, system_2, and so on. In order to improve this and to provide more meaningful names for states, the new name of the state is changed to a word or words that are in N (Object part of the input event).

After the processing presented above, a last feature of the domain information refinement is setting the destination states. This is simply done considering the source state of the next ordered quadruple as the destination state of the current ordered quadruple. However, the destination state of the last transition of the model is the first (initial) state. The FSM test criteria implemented in GTSC (DS, UIO and H-switch cover) require that the flat FSM is strongly connected. In such a machine for each pair of states (s_i, s_j), there is a path[5] connecting s_i to s_j. This explains the logic of setting the destination state of the last transition. Even though the Statecharts criteria implemented in GTSC (all-transitions, all-simple-paths, all-paths-k-C0-configuration, all-paths-k-configurations) do not have that restriction, this action makes the model translated from NL requirements more generic in the sense that a test designer may choose any of the seven GTSC test criteria to generate the test suite.

Table 7 shows the piece of the model set after the *refinement based on domain information* considering requirements SRS001 and SRS002. Assuming that N = {PDC, SWPDC, Initiation Operation Mode, OBDH, ...}, no transition is eliminated. Note that *computer* and *problem* are not in N but the Object part of the input event (after "_") of these transitions (in Table 6: *Initiation Operation Mode* in transition 5, and *OBDH* in transition 6) are in N. Precisely because the Object part of the input event is in N, the source states of transitions 2, 5 and 6 are changed according to it. Finally, the destination states are set according to the processing presented earlier.

[5] A path is a finite sequence of adjacent transitions.

Table 7 Piece of the model set after the *refinement based on domain information*. Caption: #Tr = Transition number

#Tr	Source State	Input Event	Output Event	Destination State
1	PDC	1-be 2-power on_Power Conditioning Unit	null	Initiation Operation Mode
2	Initiation Operation Mode	3-be	null	SWPDC
3	SWPDC	6-then 7-accomplish_post	null	PDC_3
4	PDC_3	9-present_irrecoverable problem	null	Initiation Operation Mode
5	Initiation Operation Mode	13-remain	null	OBDH
6	OBDH	15-not 16-be 17-propagate	null	PDC_3

4.2.2 Word sense disambiguation refinement

According to Navigli (2009), "Word Sense Disambiguation (WSD) is the ability to identify the meaning of words in context in a computational manner". In the field of Natural Language Processing, WSD has been studied for a long time. The point is that a word can have different lexical categories, e.g. "bank" can be a verb or a noun. It is the task of POS tagging to determine the correct one[6]. However, even in the same category a word can have different meanings (senses). For instance, the noun "bank" can be a depository financial institution, the building of the financial institution, or a long pile just to name a few. Thus, WSD is used to identify the correct sense in a certain context.

WordNet is an electronic lexical database created and maintained at Princeton University (Miller, 1998). The basic building block of *WordNet* is a *synset* consisting of all the words that express a given concept. Alternatively, we may say that *WordNet* encodes concepts in terms of sets of synonyms (the *synsets*) (Navigli, 2009). Hence, *WordNet*'s design resembles a thesaurus but in some aspects it also resembles a traditional dictionary, providing definitions and sample sentences for its *synsets*.

A graph-based algorithm for WSD was proposed by Sinha and Mihalcea (2007). In their approach, they constructed a sense (label) dependency graph based on measures of word semantic similarity like the *Leacock and Chodorow* (Leacock and Chodorow, 1998), *Jiang and Conrath* (Jiang and Conrath, 1997), and *Lesk* (Lesk, 1986) measures. These measures work well in the *WordNet* hierarchy. A weighted, undirected, not fully connected sense dependencies graph is derived by adding a vertex for each admissible sense of the words in a text, and an edge for each pair of senses of distinct words for which a dependency is identified. A window, wn, is defined so that no edges will be drawn between senses corresponding to words that are more than wn words apart, counting all running words, i.e. nouns, verbs, adjectives, and adverbs. The set $usefulWords$ helps to define such running words (Subsection 4.1). After the graph construction, the scores of senses are determined using some graph-based centrality algorithms like *indegree* and an adaptation of the *PageRank* algorithm. Finally, the most likely set of senses is determined by identifying the sense with the highest score for each word.

In the SOLIMVA tool, we implemented an adaptation of the above graph-based approach taking into account only one similarity measure, *Jiang and Conrath*, and one graph-based centrality algorithm, *indegree*. The goal of such adaptation was to automate the identification of the semantics

[6] POS tagging is also known as word category disambiguation.

related to the generated model. Specifically, the idea was to automatically identify self transitions in the resulting Statecharts model.

Considering the BSAO tuples and the *refinement based on domain information* (Subsection 4.2.1), the default behavior is: if the current state of the model is s_i, the next state is s_j where $i \neq j$. We would like to identify in which situations and based on the set of NL requirements the next state is s_i, i.e. when a self transition occurs within the model. To achieve this goal, we manually searched the *synsets* related to **verbs** in *WordNet* to find verb's senses which mean "remain in a same place". Our interpretation is that finding a verb with this particular sense implies that the model should exhibit a self transition.

We found 11 verbs (continue, remain, stay, etc.) with a total of 21 senses which met the desired feature. These 21 senses are precisely the elements of the $S_{TM}.F$ set. The $S_{TM}.F$ set is independent of the application domain and thus the test designer does not need to change it. A sample of the $S_{TM}.F$ is as follows, where the number indicates the sense number as defined in *WordNet*:

$S_{TM}.F$ = {remain#v#1, remain#v#2, stay#v#1, ..., rest#v#6, ...}.

In order to obtain *Jiang and Conrath* measures between pairs of verbs, we used the *Java WordNet::Similarity* (University of Sussex, 2010), a Java version of the Perl *WordNet::Similarity* package developed by the University of Minessota (Pedersen et al, 2004). Both packages use as corpus, by default, SemCor (Miller et al, 1993) which is a manually sense-tagged subset of the Brown Corpus. Besides, we are using version 2.1 of *WordNet*.

Among the four graph-based centrality algorithms used by Sinha and Mihalcea (2007), we chose and implemented the *indegree* algorithm within the SOLIMVA tool. For an undirected weighted graph $G = (V, E)$ where V is the set of vertices and E is the the set of edges, the indegree is defined as:

$$indegree(V_a) = \sum_{V_b \in V} wg_{ab}$$

where wg_{ab} is the weight on the edge between V_a and V_b. In other words, the *indegree* of a vertex V_a is obtained taking into account the weights on the edges, and adding them together into a score. In our case, V_a and V_b are senses of two distinct verbs, and wg_{ab} is the *Jiang and Conrath* measure between these two verb's senses. Furthermore, we selected $wn = 4$ which was the value that provided the best results regarding the correct word sense assignment. In order to identify the sense of a verb, the algorithm only figures out the sense with the highest score among all the senses of a verb.

We decided to use only the *Jiang and Conrath* similarity measure because our goal was to disambiguate the senses of verbs. According to the results shown in Sinha and Mihalcea (2007), the best measure in terms of true positives for verbs was the *Leacock and Chodorow* followed closely by the *Jiang and Conrath* measure (the *Jiang and Conrath* measure was 4.55% worse than the *Leacock and Chodorow* measure; other measures were more than 10% worse than the *Leacock and Chodorow* measure). At first, we tried the *Leacock and Chodorow* measure in the SWPDC's Software Requirements Specification but the results were not very promising. Thus, we selected the *Jiang and Conrath* measure which presented a better performance.

The reasoning behind the selection of the *indegree* graph-based centrality algorithm was also because this was the best algorithm for verb sense disambiguation outperforming the other three algorithms (Sinha and Mihalcea, 2007). The initial goal was to implement the four graph-based centrality algorithms and combine them in a voting scheme as proposed by Sinha and Mihalcea (2007). But, the results presented for verbs according to their approach were not very promising and we decided to implement the algorithm with best performance, the *indegree*.

The main WSD refinement algorithm is shown in Fig. 9. The verbs for WSD set (*vrbwsd*) contains all verbs of the set of NL requirements that match the verbs defined in $S_{TM}.F$. Each ordered pair of *vrbwsd* is composed of: the key ("counter of Useful Words ($cntUW$) - word") defined in Subsection 4.1 (upper index ivr), and one of two possible values (upper index inf), "self" and "no". This set has been previously generated (Fig. 7) which informs whether there are verbs within the NL requirements that characterize a self transition in the resulting model (value "self"). Hence, our adaptation of the approach proposed by Sinha and Mihalcea (2007) was in fact implemented as part of the BSAO tuples generation algorithm. However, to make use of the results by adapting the Sinha and Mihalcea (2007) approach to identify self transitions in the Statecharts model, we need additional steps as described in Fig 9. Finally, the importance of the key is due to the fact that naturally the same verb (for instance, stay) may occur several times within the NL requirements indicating or not indicating the presence of several self transitions. Hence, we need to know which word (verb) we are precisely analyzing.

The algorithm in Fig. 9 takes the *vrbwsd* set as input, and first extracts the Action part (if any) of the input event of a transition and realizes whether it matches a key of *vrbwsd* (lines 7 and 8). Assuming this is true and also that the information related to the key matches the value "self" (line 10), the algorithm performs a backward search until it finds a previous source state which is the same as the current source state where the verb (key) characterizes a self transition (lines 12 to 20). As long as this does not occur, the input events of all transitions backward traversed are stored ($tempIET$, line 14) in order to compose the new input event of a transition in the model (line 24). The new current destination state is exactly the current source state which is the behavior we expect in such a situation (line 25). Intermediate and needless states and transitions are removed from the model (line 26), and the new next source state is also set (line 27).

To sum up, there are four conditions to be satisfied so that a self transition is detected and added in the model after the WSD refinement. First, the verb must match the verbs in $S_{TM}.F$. Second, the information must be "self". Third, a previous source state must match the current source state in the backward search. This is important because if there is no match we will not be able to decide which is the source state of the self transition. Finally, during the backward search, none of the input events must match an element of $R.I_E$. Again, this demonstrates the priority of reactiveness over other behaviors in the resulting model.

Table 8 shows the piece of the model set after the WSD refinement based on requirements SRS001 and SRS002. Note that due to the input event *13-remain* (Table 7), and the satisfaction of all previously mentioned conditions, transition number 2 is a self transition. Also note that the input event of transition 2 is the concatenation of previous input events as designed in the WSD refinement algorithm.

4.2.3 Refinement for inclusion of hierarchy

The output model obtained after the WSD refinement is no more than an FSM as defined by Petrenko and Yevtushenko (2005). In order to obtain a Statecharts model, we created a strategy to incorporate hierarchy (depth) into the final model. This is the last automated refinement proposed within the SOLIMVA tool.

The algorithm that we designed and implemented partitions the set of states of the model based on information gathered from the $S_{TM}.Y$ function. Such a function is a mapping among some names of source states (*src*) and input events (*iev*) of transitions, which are elements of the input pattern set ($Y.I_P$), and names that will define the COMPOSITE states of the Statecharts model, which are

input : model $mod = \{m_x^y \,|\, y = src, iev, oev, des\}, x = 1..zi$
input : dictionary $dic = \{N, R, S_{TM}.C, S_{TM}.F, S_{TM}.Y\}$
input : verbsWSD $vrbwsd = \{v_p^q \,|\, q = ivr, inf\}, p = 1..r$
output: model $mod = \{m_x^y \,|\, y = src, iev, oev, des\}, x = 1..zo$

```
1  inpAct, newIET, tempIET ← empty;
2  matIET, matSrc ← false;
3  ib, it ← 1;
4  for p ← 1 to r do
5  |   x ← 1;
6  |   while x ≤ zi ∧ ¬matIET do
7  |   |   inpAct ← extractAction(mod);
8  |   |   if v_p^{ivr} = inpAct then
9  |   |   |   matIET ← true;
10 |   |   |   if v_p^{inf} = "self" then
11 |   |   |   |   ib ← x;
12 |   |   |   |   while ib ≥ 1 ∧ ¬matSrc do
13 |   |   |   |   |   ib ← ib − 1;
14 |   |   |   |   |   tempIET_{it} ← m_{ib}^{iev};
15 |   |   |   |   |   it ← it + 1;
16 |   |   |   |   |   if m_x^{src} = m_{ib}^{src} ∧ m_{ib}^{iev} ∉ R.I_E then
17 |   |   |   |   |   |   newIET ← reversetempIETAndGeneratenewIET(tempIET);
18 |   |   |   |   |   |   matSrc ← true;
19 |   |   |   |   |   end
20 |   |   |   |   end
21 |   |   |   |   tempIET ← empty;
22 |   |   |   |   it ← 1;
23 |   |   |   |   if matSrc then
   |   |   |   |   |   // new current Input Event Transisiton.
24 |   |   |   |   |   m_{ib}^{iev} ← newIET;
   |   |   |   |   |   // new current Destination State.
25 |   |   |   |   |   m_{ib}^{des} ← m_{ib}^{src};
   |   |   |   |   |   // remove unnecessary states and transitions.
26 |   |   |   |   |   mod ← removeStatesTransitions(mod, ib + 1);
   |   |   |   |   |   // new next Source State.
27 |   |   |   |   |   m_{ib+1}^{src} ← m_{ib}^{src};
28 |   |   |   |   |   matSrc ← false;
29 |   |   |   |   |   newIET ← empty;
30 |   |   |   |   end
31 |   |   |   end
32 |   |   end
33 |   |   x ← x + 1;
34 |   end
35 |   matIET ← false;
36 end
```

Fig. 9 Main WSD refinement algorithm

Table 8 Piece of the model set after the WSD refinement. Caption: #Tr = Transition number

#Tr	**Source State**	**Input Event**	**Output Event**	**Destination State**
1	PDC	1-be 2-power on_Power Conditioning Unit	null	Initiation Operation Mode
2	Initiation Operation Mode	3-be#6-then 7-accomplish_post# 9-present_irrecoverable problem#	null	Initiation Operation Mode
3	Initiation Operation Mode	15-not 16-be 17-propagate	null	PDC_3

elements of the output pattern set ($Y.O_P$). Note that the selected names of source states are certainly in N as well as the names of input events are in $R.I_E$.

One basic assumption is that no COMPOSITE state will be created if a self transition exists in a certain state. Besides, the general idea is to detect whether the *src* and *iev* of transitions match some element of $Y.I_P$ and, if so names of appropriate source and destination states are changed so that depth can be incorporated in the model.

Some other requirements were considered when designing the algorithm for incorporating hierarchy. First, we decided that the depth was limited to two levels, say the main level and a second hierarchy level. Although depth is an important feature of Statecharts, from the point of view of models for system and acceptance test case generation, too many hierarchy levels may make the entire modeling difficult to read. Our experience is that two levels are enough for this purpose.

A COMPOSITE state is created right from a state m_x^{src} if the aforementioned match considering $Y.I_P$ occurs, and also if at least in the next two transitions there is no match regarding *src* and *iev*, i.e. if $m_{x+1}^{src}, m_{x+1}^{iev}, m_{x+2}^{src}, m_{x+2}^{iev}$ do not match elements in $Y.I_P$. In doing this, we avoid creating an excessive number of COMPOSITE states and/or COMPOSITE states embodying few states.

There is no priority among COMPOSITE states. In other words, at any time the behavior may be "leave the COMPOSITE state m_x^{src}" if the conditions previously described, related to the match, are satisfied. Another feature is that the initial state of the main model is not allowed to make part of a COMPOSITE state, and there is also a mechanism to include the *In State* conditions of Statecharts into the final model.

Let us consider the SWPDC case study. Hence, $Y.I_P$, $Y.O_P$, and $S_{TM}.Y$ can be composed of:

$Y.I_P$ = {Safety Operation Mode, CH-OP-MODE-NOMINAL, ... },
$Y.O_P$ = {Safety Operation Mode, Nominal Operation Mode, ... },
$S_{TM}.Y$ = {(Safety Operation Mode, Safety Operation Mode), (CH-OP-MODE-NOMINAL, Nominal Operation Mode), ... }.

5 Application of the SOLIMVA methodology

This section presents in detail the application of the SOLIMVA methodology/tool using as case study the SWPDC software product (Santiago et al, 2007). The test designer starts by defining the Dictionary. Then, he/she can create the Name (N) set and Reactiveness function (R) in accordance with Table 9, and Table 10 shows a possibility for the *Semantic Translation Model*: Control ($S_{TM}.C$) set, Self Transition ($S_{TM}.F$) set, and Hierarchy ($S_{TM}.Y$) function. In functions, the representation is $x \rightarrow y$ where x is an element of the domain of the function and y is an element of the codomain of the

function. Furthermore, the elements in CAPITAL letters of the domain ($R.I_E$) and codomain ($R.O_E$) of R are simply abbreviations for commands and responses of the PDC-OBDH Communication Protocol Specification. Thus, VER-OP-MODE is an abbreviation for the command VERIFY PDC's OPERATION MODE. As previously stated, $S_{TM}.C$ and $S_{TM}.F$ are already defined in configuration files of the SOLIMVA tool and hence the user does not need to alter them. However, if necessary the user has an option to change them.

Table 9 Sample of the Name set and the Reactiveness function for the SWPDC case study

Name	***Reactiveness***
PDC	VER-OP-MODE → INFO-OP-MODE
SWPDC	PREP-HK → CMD-REC
Initiation Operation Mode	TX-DATA-SCI-End → SCI-DATA or NO-DATA
Safety Operation Mode	CH-OP-MODE-NOMINAL → CMD-REC
Nominal Operation Mode	CH-OP-MODE-SAFETY → CMD-REC
EPP Hx	Several TX-DATA-HK → Several HK-DATA or NO-DATA
OBDH	...
...	

Table 10 Sample of the *Semantic Translation Model* for the SWPDC case study

Control	***Self Transition***	***Hierarchy***
if	remain#v#1	Initiation Operation Mode → Initiation Operation Mode
...	remain#v#2	Safety Operation Mode → Safety Operation Mode
	stay#v#1	Nominal Operation Mode → Nominal Operation Mode
	stay#v#2	CH-OP-MODE-NOMINAL → Nominal Operation Mode
	stay#v#4	CH-OP-MODE-SAFETY → Safety Operation Mode
	rest#v#1	...
	rest#v#6	
	continue#v#1	
	...	

After this, scenarios are defined using the strategy described in Section 3. First, the core elements in this case study are the 37 commands that the OBDH can send to PDC which are defined in the PDC-OBDH Communication Protocol Specification. These 37 commands were grouped into 28 scenarios. These are very simple scenarios consisting essentially in switching the PDC on and send these commands in order to realize whether PDC correctly receives and processes such commands.

For the normal scenarios, Table 11 shows a possible choice of factors and levels for the SWPDC case study. The meanings of the factors Cmd, OpMode, and Services as well as the levels HwSwHnd, DtAcqTx, Nom, Sci, and Hk have already been given in Section 3. The explanation for the remaining factors and levels follows:

1. Levels of the Cmd factor: processing activities to manage PDC's operation mode (OpMMgm), load new program into PDC's Data Memory on the fly (PrLoad);
2. Levels of the OpMode factor: Initiation/Initialization (Init), Safety (Safe), and Diagnosis (Diag) are other PDC's operation modes;

3. Levels of the Services factor: Services related to acquisition, formatting, and transmission of Test (Tst) and Diagnosis (Dg) Data, generation, formatting, and transmission of Dump Data (Dmp), loading new program into PDC's Data Memory on the fly (Load);
4. StartMode factor. This factor relates to the way PDC is started: Power On (PwrOn) or Reset (Reset).

Table 11 Factors and levels for the SWPDC case study

Factors	**Levels**						
Cmd	HwSwHnd	OpMMgm	DtAcqTx	PrLoad	Inv		
OpMode	Nom	Init	Safe	Diag	Inv		
Services	Sci	Hk	Dmp	Load	Dg	Tst	Inv
StartMode	PwrOn	Reset	Inv				

Since there are four factors and as explained in the *Define Scenarios* activity (Fig. 4), then *strength* = 3. The combinatorial designs algorithm produced 175 factor combinations which shall be interpreted to derive 175 normal scenarios. One example is the factor combination 71: {DtAcqTx, Nom, Sci, Inv}. However, the test designer may neglect the level *Inv* assuming that robustness will be covered by another factor combination. Hence, factor combination 71 becomes: {DtAcqTx, Nom, Sci, -}. The interpretation of such factor combination defines scenario 71 to "acquire, format, and transmit Scientific Data in the Nominal Operation Mode".

We analyzed all normal scenarios to see if they needed to be unfolded. Normal scenarios 73 ({DtAcqTx, Nom, Dmp, -}) whose interpretation is "generation, formatting, and transmission of Dump Data in the Nominal Operation Mode", and 94 ({DtAcqTx, Diag, Dmp, -}) whose interpretation is "generation, formatting, and transmission of Dump Data in the Diagnosis Operation Mode", are two normal scenarios to be unfolded. The Dump Data (Dmp) service refers to get data from pieces of PDC's Program or Data Memory. Organization of PDC's Program Memory is simple with a 64-kByte program address space, from 0000h to FFFFh (h = hexadecimal). However, organization of PDC's Data Memory is more complicated as shown in Fig. 10. The size of the data address space is also 64 kBytes, from 0000h to FFFFh. However, there is a paging mechanism which enables PDC to access more than 64 kBytes of Data Memory. Each one of the 8 pages in Fig. 10 has a size of 32 kBytes. Hence, in order to dump data from PDC's memories the user must:

- select the memory from where to dump data. The options are Program and Data Memories. In case the user selects the Data Memory, the page (0 to 7) must be selected too;
- provide the initial and final 16-bit memory addresses.

The selection is accomplished by means of a specific command defined in the PDC-OBDH Communication Protocol Specification. Then, we can define dump from different pieces of memory areas, and we may also emphasize robustness aspects of SWPDC; for instance, what is its behavior when the initial address is greater than the final address, the initial or final address is less than the minimum physical address allowed for a certain type of memory and so on. These facts explain the need for unfolding.

Table 12 shows the configuration of factors and levels for unfolding normal scenarios 73 and 94. The priority factor in this case is the type of Memory (Mem; Prg = Program Memory, DtP0 = Data Memory - Page 0, ..., DtP7 = Data Memory - Page 7) from where the dump will take place, and

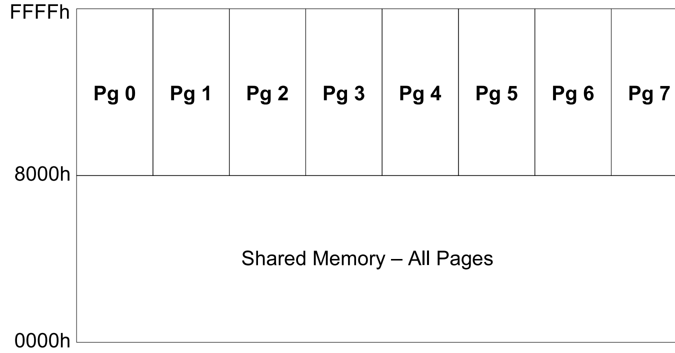


Fig. 10 Organization of PDC's Data Memory. Caption: Pg = Page

strength = 2. The initial memory address (IniAdd) and final memory address (FinalAdd) are the other factors. Since the priority factor has 10 levels, then not only normal scenario 73 but also normal scenario 94 will be replaced with 10 unfolded scenarios each (note that the only difference between normal scenarios 73 and 94 is the PDC's operation mode). These unfolded scenarios are identified as 73.1, 73.2, ..., 73.10, 94.1, 94.2, ..., 94.10.

Table 12 Unfolding normal scenarios 73 and 94

Factors	**Levels**									
Mem	Prg	DtP0	DtP1	DtP2	DtP3	DtP4	DtP5	DtP6	DtP7	Inv
IniAdd	InRng	Min	Max	LessMin	GreatMax					
FinalAdd	InRng	Min	Max	LessMin	GreatMax					

Running the combinatorial designs algorithm, the factor combinations that are interpreted for deriving each unfolded scenario are shown in Table 13. As mentioned in Section 3, the test designer adds NL requirements to characterize each unfolded scenario according to the directives provided by the combinatorial designs algorithm.

It is interesting to stress the focus on Robustness testing related to this unfolding process. For instance, unfolded scenarios 73.3 or 94.3 propose the following situations, where all cases relate to Page 1 of PDC's Data Memory (DtP1):

- IniAdd = Min, FinalAdd = LessMin;
- IniAdd = Max, FinalAdd = InRng;
- IniAdd = LessMin, FinalAdd = Min;
- IniAdd = GreatMax, FinalAdd = Min;
- IniAdd = Min, FinalAdd = GreatMax;
- IniAdd = InRng, FinalAdd = Max.

SWPDC must process only the last situation (IniAdd = InRng, FinalAdd = Max) because all the others have innapropriate settings of initial and/or final memory addresses. The unfolding process

Table 13 Factor combinations for deriving unfolded scenarios from normal scenarios 73 and 94. Caption: Min/Max = Minimum/Maximum allowed memory address; LessMin/GreatMax = Less/Greater than Minimum/Maximum allowed memory address; InRng = Address between Min and Max (In Range)

Unfolded Scenario	**Factor Combinations**
73.1 and 94.1	{Prg, InRng, InRng}, {Prg, Min, Min}, {Prg, Max, Max}, {Prg, LessMin, LessMin}, {Prg, GreatMax, GreatMax}
73.2 and 94.2	{DtP0, InRng, Min}, {DtP0, Min, InRng}, {DtP0, Max, LessMin}, {DtP0, LessMin, Max}, {DtP0, GreatMax, InRng}, {DtP0, Min, GreatMax}
73.3 and 94.3	{DtP1, InRng, Max}, {DtP1, Min, LessMin}, {DtP1, Max, InRng}, {DtP1, LessMin, Min}, {DtP1, GreatMax, Min}, {DtP1, Min, GreatMax}
73.4 and 94.4	{DtP2, InRng, LessMin}, {DtP2, Min, Max}, {DtP2, Max, Min}, {DtP2, LessMin, InRng}, {DtP2, GreatMax, Max}, {DtP2, Min, GreatMax}
73.5 and 94.5	{DtP3, InRng, GreatMax}, {DtP3, Min, InRng}, {DtP3, Max, Min}, {DtP3, LessMin, Max}, {DtP3, GreatMax, LessMin}
73.6 and 94.6	{DtP4, InRng, InRng}, {DtP4, Min, GreatMax}, {DtP4, Max, Min}, {DtP4, LessMin, Max}, {DtP4, GreatMax, LessMin}
73.7 and 94.7	{DtP5, InRng, InRng}, {DtP5, Min, Min}, {DtP5, Max, GreatMax}, {DtP5, LessMin, Max}, {DtP5, GreatMax, LessMin}
73.8 and 94.8	{DtP6, InRng, InRng}, {DtP6, Min, Min}, {DtP6, Max, Max}, {DtP6, LessMin, GreatMax}, {DtP6, GreatMax, LessMin}
73.9 and 94.9	{DtP7, InRng, InRng}, {DtP7, Min, Min}, {DtP7, Max, Max}, {DtP7, LessMin, LessMin}, {DtP7, GreatMax, GreatMax}
73.10 and 94.10	{Inv, InRng, InRng}, {Inv, Min, Min}, {Inv, Max, Max}, {Inv, LessMin, LessMin}, {Inv, GreatMax, GreatMax}

generated 20 additional scenarios, 10 to replace normal scenario 73 and another 10 to replace normal scenario 94.

Another four normal scenarios needed to be unfolded: normal scenario 109 ({PrLoad, Nom, Load, -}; "loading new program into PDC's Data Memory on the fly in the Nominal Operation Mode") which contributed with 6 additional scenarios; normal scenarios 2 ({HwSwHnd, Nom, Hk, -}), 16({HwSwHnd, Safe, Hk, -}), 23 ({HwSwHnd, Diag, Hk, -}) whose interpretations are "verification of correct implementation of commands related to software parameters manipulation, and generation, formatting, and transmission of Housekeeping Data in the Nominal (scenario 2), Safety (scenario 16) or Diagnosis (scenario 23) Operations Mode", and that provided 7 additional scenarios each one, adding 21 new scenarios. Hence, the total number of scenarios proposed by the SOLIMVA methodology was 244.

For each scenario, a set of NL requirements is chosen. As a matter of illustration, we will show the set of NL requirements in order to characterize normal scenario 71: {DtAcqTx, Nom, Sci, -}. Table 14 shows the selected NL requirements.

After the selection of the requirements, the next step is to generate the Statecharts model. Fig. 11 shows the main model generated by the SOLIMVA tool for normal scenario 71. Note the self transition in state *Initiation Operation Mode*. This behavior occurs precisely due to the verb "remain" in requirement SRS002. The WSD refinement identified the sense of "remain" as one of the senses in $S_{TM}.F$. There are three COMPOSITE states (symbol ∞) in the main Statecharts model: *Nominal Operation Mode* (Fig. 12), *Safety Operation Mode* (Fig. 13), and *Safety Operation Mode_2*. When omitted, the output event within a transition is *null*.

In the sequence, GTSC is run to generate the Abstract Test Cases. The Abstract Test Suites based on all-transitions, all-simple-paths, and all-paths-k-C0-configuration test criteria are presented

Table 14 Set of NL requirements that characterize normal scenario 71. Caption: Req = Requirement; Id = Identification

Req Id	Req Description
SRS001	The PDC shall be powered on by the Power Conditioning Unit.
SRS002	The PDC shall be in the Initiation Operation Mode after being powered on. The SWPDC shall then accomplish a POST. If PDC presents any irrecoverable problem, this computer shall remain in the Initiation Operation Mode and such a problem shall not be propagated to the OBDH.
SRS003	If PDC does not present any irrecoverable problem, after the initiation process, the PDC shall automatically enter into the Safety Operation Mode.
POCP001	The PDC can only respond to requests (commands) from OBDH after the PDC has been energized for at least 1 minute. If OBDH sends commands within less than 1 minute, the OBDH shall not receive any response from PDC.
RB001	The OBDH shall send VER-OP-MODE to PDC.
RB002	The PDC shall switch each Event Pre-Processor (EPP Hx, x = 1 or 2) on or off independently, when the OBDH sends distinct commands to perform such actions.
PECP001	Each EPP Hx can only respond to requests (commands) from PDC after each EPP Hx has been energized for at least 30 seconds. If PDC sends commands within less than 30 seconds to a certain EPP Hx, the PDC shall not receive any response from this EPP Hx.
SRS004	The OBDH should wait 600 seconds before asking for a Housekeeping Data frame.
SRS005	Housekeeping data transmission shall start with PREP-HK. After that, the OBDH can send several TX-DATA-HK to PDC. The transmission shall be ended with TX-DATA-SCI-End.
RB003	The OBDH shall send CH-OP-MODE-NOMINAL to PDC.
RB001	The OBDH shall send VER-OP-MODE to PDC.
POCP002	The OBDH should wait 10 seconds before asking for a Scientific Data frame.
SRS006	The SWPDC shall obtain and handle scientific data from each EPP Hx. The SWPDC shall also accept scientific data transmission requests from OBDH.
RB004	The OBDH shall send CH-OP-MODE-SAFETY to PDC. After that, the PDC shall be in the Safety Operation Mode.
RB002	The PDC shall switch each Event Pre-Processor (EPP Hx, x = 1 or 2) on or off independently, when the OBDH sends distinct commands to perform such actions.
RB005	After switching both EPPHxs off via PDC, the OBDH shall switch the PDC off via the Power Conditioning Unit.

in Table 15. An Abstract Test Case is composed of a set of *input / output* pairs, and it is delimited by left and right braces.

The Abstract Test Cases must be translated into the Executable Test Cases in order to stimulate the IUT. Table 16 shows examples of such translations considering the second Abstract Test Case of the all-transitions Abstract Test Suite. It is interesting to note that in some cases, there is no specific test input data to stimulate the IUT but only actions shall be performed and certain behaviors shall occur. Besides, the mapping is not one to one, i.e. not necessarily one *input / output* pair of the Abstract Test Case will derive exactly one *test input data / expected result* of the Executable Test Case.

Another remark is that most of the translations shown in Table 16 are applied not only to other Abstract Test Suites derived from other test criteria for normal scenario 71 but also to other Abstract Test Suites due to the all-transitions or other test criteria regarding another scenario. The point is that it is not necessary to do all the mapping when considering other scenarios: most of the translations are reusable.

The complete translation of the three Abstract Test Cases that compose the Abstract Test Suite derived from the all-transitions test criterion for normal scenario 71 into the Executable Test Suite is shown in Table 17. *Action1*, *Action2*, *Action3*, *Action4*, and *Action5* are defined in Table 16. *Action6*

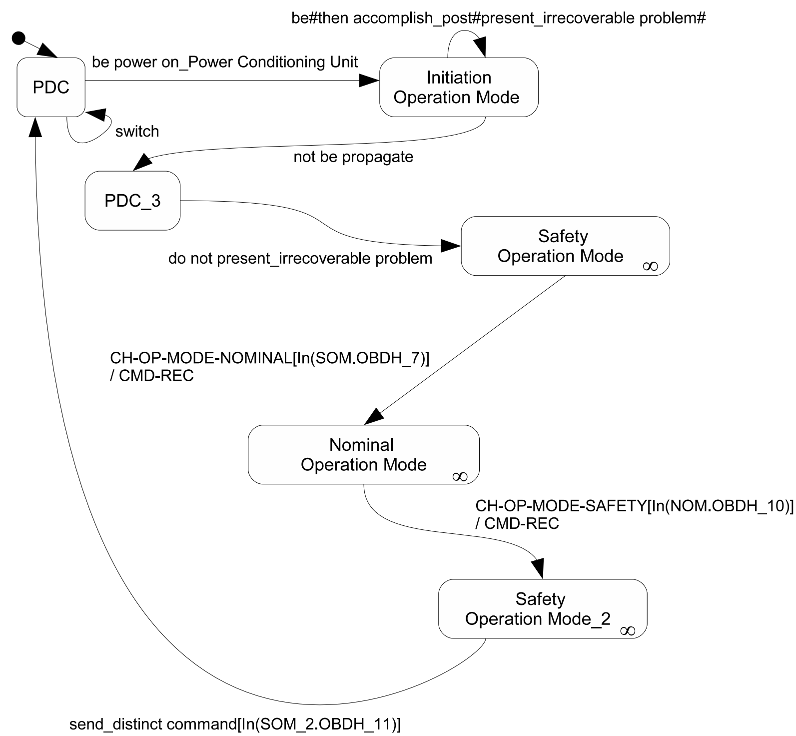


Fig. 11 The main Statecharts model derived from NL requirements that characterize normal scenario 71

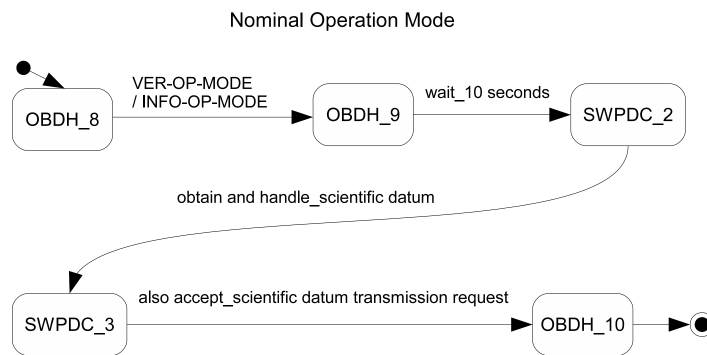


Fig. 12 Normal scenario 71: COMPOSITE state *Nominal Operation Mode*

is the translation of the test step *be#then accomplish_post#present_irrecoverable problem#/null*.

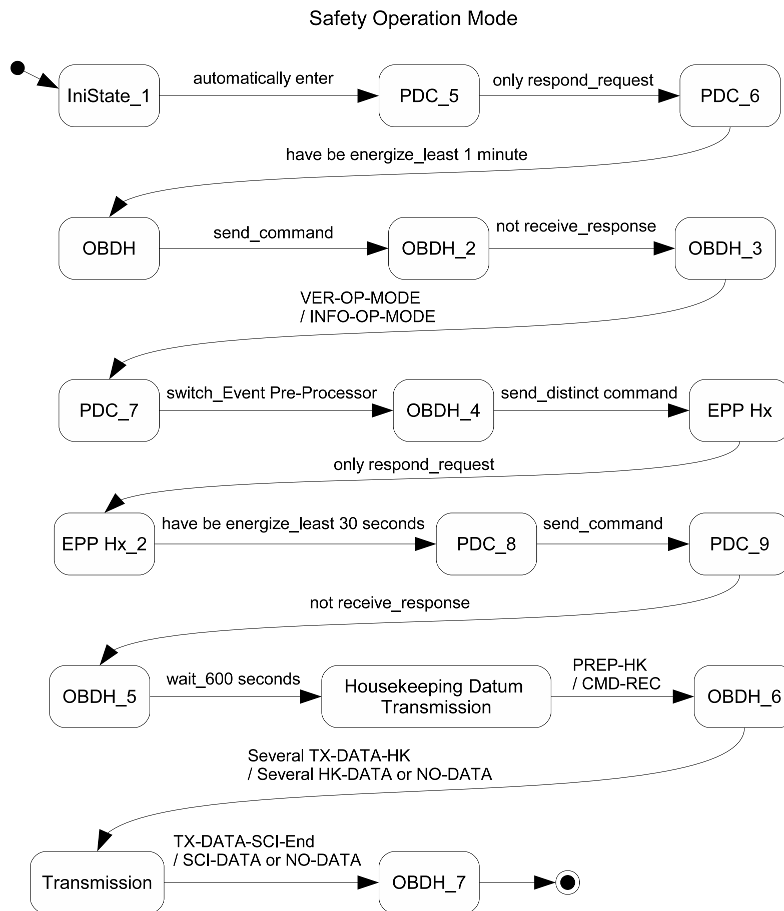


Fig. 13 Normal scenario 71: COMPOSITE state *Safety Operation Mode*

Hence, *Action6* directs the test designer to simulate an irrecoverable problem during the intitiation (initialization) process of PDC and realize whether PDC remains in the *Initiation Operation Mode*, as presented in Fig. 11. *Action7* is to switch PDC off. The idea then is just to substitute one or more *input / output* pairs of the Abstract Test Suite for the corresponding *test input data / expected result* of the Executable Test Suite as shown in Table 16. Translations into other Executable Test Suites due to other test criteria follow the same principles.

The concrete *test input data / expected result* of the Executable Test Cases are obtained just by replacing the abbreviations of commands/responses with the values as specified in the PDC-OBDH Communication Protocol Specification. For instance, the first VER-OP-MODE / INFO-OP-MODE (Table 17) is substituted for (all values are in hexadecimal):

Table 15 Abstract Test Suites for normal scenario 71

Test Criterion	Abstract Test Cases
all-transitions	{be power on_Power Conditioning Unit/null, be#then accomplish_post#present_irrecoverable problem#/null}, {be power on_Power Conditioning Unit/null, not be propagate/null, do not present_irrecoverable problem/null, automatically enter/null, only respond_request/null, have be energize_least 1 minute/null, send_command/null, not receive_response/null, VER-OP-MODE/INFO-OP-MODE, switch_Event Pre-Processor/null, send_distinct command/null, only respond_request/null, have be energize_least 30 seconds/null, send_command/null, not receive_response/null, wait_600 seconds/null, PREP-HK/CMD-REC, Several TX-DATA-HK/Several HK-DATA or NO-DATA, TX-DATA-SCI-End/SCI-DATA or NO-DATA, CH-OP-MODE-NOMINAL/CMD-REC, VER-OP-MODE/INFO-OP-MODE, wait_10 seconds/null, obtain and handle_scientific datum/null, also accept_scientific datum tansmission request/null, CH-OP-MODE-SAFETY/CMD-REC, be/null, switch_Event Pre-Processor/null, send_distinct command/null}, {switch/null}
all-simple-paths	{be power on_Power Conditioning Unit/null, not be propagate/null, do not present_irrecoverable problem/null, automatically enter/null, only respond_request/null, have be energize_least 1 minute/null, send_command/null, not receive_response/null, VER-OP-MODE/INFO-OP-MODE, switch_Event Pre-Processor/null, send_distinct command/null, only respond_request/null, have be energize_least 30 seconds/null, send_command/null, not receive_response/null, wait_600 seconds/null, PREP-HK/CMD-REC, Several TX-DATA-HK/Several HK-DATA or NO-DATA, TX-DATA-SCI-End/SCI-DATA or NO-DATA, CH-OP-MODE-NOMINAL/CMD-REC, VER-OP-MODE/INFO-OP-MODE, wait_10 seconds/null, obtain and handle_scientific datum/null, also accept_scientific datum tansmission request/null, CH-OP-MODE-SAFETY/CMD-REC, be/null, switch_Event Pre-Processor/null, send_distinct command/null}, {switch/null}
all-paths-k-C0	{be power on_Power Conditioning Unit/null, be#then accomplish_post#present_irrecoverable problem#/null, not be propagate/null, do not present_irrecoverable problem/null, automatically enter/null, only respond_request/null, have be energize_least 1 minute/null, send_command/null, not receive_response/null, VER-OP-MODE/INFO-OP-MODE, switch_Event Pre-Processor/null, send_distinct command/null, only respond_request/null, have be energize_least 30 seconds/null, send_command/null, not receive_response/null, wait_600 seconds/null, PREP-HK/CMD-REC, Several TX-DATA-HK/Several HK-DATA or NO-DATA, TX-DATA-SCI-End/SCI-DATA or NO-DATA, CH-OP-MODE-NOMINAL/CMD-REC, VER-OP-MODE/INFO-OP-MODE, wait_10 seconds/null, obtain and handle_scientific datum/null, also accept_scientific datum tansmission request/null, CH-OP-MODE-SAFETY/CMD-REC, be/null, switch_Event Pre-Processor/null, send_distinct command/null}

VER-OP-MODE: EB 80 00 00 00 00 04 00 00 00 FE 91;
INFO-OP-MODE: EB 20 XX XX XX XX 83 00 00 01 01 XX XX.

In the expected result, XX represents values that are very difficult to predict in advance such as the time stamp that indicates the exact value of the clock of the PDC computer when PDC is assembling a response to be sent back to OBDH. In these cases, we do not need to worry about these values.

5.1 Comparing SOLIMVA with an expert approach

This subsection compares the models and Executable Test Cases generated by the SOLIMVA methodology/tool with others manually generated by a test designer who is an expert in the SWPDC software product. The 20 models generated by the expert can be seen in Santiago Júnior et al (2010).

Table 16 Examples of the translation from the Abstract Test Suite into the Executable Test Suite

Test Step	**Test Input Data**	**Expected Result**	**Action**
be power on_Power Conditioning Unit/null	Action1	-	Action1: Switch PDC on
not be propagate/null + do not present_irrecoverable problem/null + automatically enter/null	Action2	-	Action2: Realize whether the PDC is in the Safety Operation Mode after the initiation process
only respond_request/null + have be energize_least 1 minute/null + send_command/null + not receive_response/null	VER-OP-MODE Action3	Timeout	OBDH shall send any command within less than 1 minute since the PDC has been energized. Action3: OBDH should wait 60 seconds to send a command
VER-OP-MODE/INFO-OP-MODE	VER-OP-MODE	INFO-OP-MODE	OBDH shall send this command
only respond_request/null + have be energize_least 30 seconds/null + send_command/null + not receive_response/null	PREP-TST Action4 TX-DATA-TST TX-DATA-TST TX-DATA-SCI	CMD-REC NO-DATA NO-DATA NO-DATA	OBDH shall send/perform all these commands/action within less than 30 seconds since each EPP Hx has been energized. Hence, PDC will try to get Test (TST) Data from each EPP Hx but a timeout will occur in the PDC side of the communication. The result is that no Test Data frame (NO-DATA) will be sent to the OBDH. Action4: OBDH should wait 10 seconds to send a command
wait_600 seconds/null	Action5	-	Action5: OBDH should wait 600 seconds to send a command
TX-DATA-SCI-End/SCI-DATA or NO-DATA	TX-DATA-SCI	NO-DATA	OBDH shall send this command
wait_10 seconds/null	Action4	-	Action4: OBDH should wait 10 seconds to send a command
obtain and handle_scientific datum/null + also accept_scientific datum transmission request/null	TX-DATA-SCI	SCI-DATA	After waiting for 10 seconds, the OBDH can send this command. In case that many Scientific Data frames shall be requested, this process is repeated as many times as needed (wait 10 s, send the command)

The goal of such a comparison was two-fold: (i) we would like to know if the SOLIMVA methodology is able to cover the test objectives of the set of scenarios manually developed by an expert with respect to a certain IUT; (ii) we would like to verify whether the Executable Test Cases generated according to the SOLIMVA methodology have the same characteristics of the test cases developed according to the expert's models.

Table 17 Executable Test Suite translated from the all-transitions Abstract Test Suite

Test Criterion	Executable Test Cases
all-transitions	{Action1, Action6}, {Action1, Action2, VER-OP-MODE/Timeout, Action3, VER-OP-MODE/INFO-OP-MODE, ACT-HW-EPPH1ON/CMD-REC, ACT-HW-EPPH2ON/CMD-REC, PREP-TST/CMD-REC, Action4, TX-DATA-TST/NO-DATA, TX-DATA-TST/NO-DATA, TX-DATA-SCI/NO-DATA, Action5, PREP-HK/CMD-REC, TX-DATA-HK/HK-DATA, TX-DATA-SCI/NO-DATA, CH-OP-MODE-NOMINAL/CMD-REC, VER-OP-MODE/INFO-OP-MODE, Action4, TX-DATA-SCI/SCI-DATA, Action4, TX-DATA-SCI/SCI-DATA, Action4, TX-DATA-SCI/SCI-DATA, Action4, TX-DATA-SCI/SCI-DATA, Action4, TX-DATA-SCI/SCI-DATA, Action4, TX-DATA-SCI/SCI-DATA, Action4, TX-DATA-SCI/SCI-DATA, Action4, TX-DATA-SCI/SCI-DATA, Action4, TX-DATA-SCI/SCI-DATA, Action4, TX-DATA-SCI/SCI-DATA, CH-OP-MODE-SAFETY/CMD-REC, VER-OP-MODE/INFO-OP-MODE, ACT-HW-EPPH1OFF/CMD-REC, ACT-HW-EPPH2OFF/CMD-REC}, {Action7}

Table 18 shows the comparison. The leftmost column shows the *Test Objectives* associated to a scenario, *Expert* shows the number of the manually generated scenario, and the *SOLIMVA* column shows the corresponding factor combinations whose interpretations derive the scenarios which have associated the same test objectives of the manual (expert) scenarios. The number of the scenario in the *SOLIMVA* column refers to those generated only by the main execution of the combinatorial designs algorithm, i.e. one that resulted in 175 normal scenarios.

The first goal of our comparison was achieved. All test objectives of the expert's scenarios were satisfactorily covered by SOLIMVA's scenarios. Note that a single expert's scenario can be addressed by more than one corresponding SOLIMVA's scenario. For instance, the test objective associated to expert's scenario 1 can be addressed by any of the following SOLIMVA's scenarios: 43, 44, 45, 46, 47, 48, 49. Besides, a factor combination may be interpreted in different ways. In Table 18, three different test objectives are related to factor combination 16: "Switching EPP Hxs on and off", "Changing software parameters in the Safety Operation Mode", and "Process of Power On". It is up to the test designer to choose one of these test objectives for SOLIMVA's scenario 16, and leave the others to be addressed by interpreting other factor combinations.

Besides covering all the test objectives, the SOLIMVA philosophy also pointed out some problems in the expert's strategy. Expert's scenario 4 could be broken into two different scenarios: one addressing the process of power on and another related to the reset of PDC. As expert's scenario 4 can be mapped to four distinct groups of scenarios in SOLIMVA, this demonstrates that the test objectives of the expert were confused trying to deal with many different aspects into a single scenario. Fig. 14 shows the main Statecharts model derived by the expert for scenario 4. Note that this model is an AND state with three parallel substates: *Initiation*, *Timing*, and *PowerState*. Let the sequence of transitions be (only the input events are shown): {*switchPDCOn, POSTOk, tsinc, VER_OP_MODE[In (Power)]*}, where *tsinc* is an internal event. After such a sequence of transitions, the COMPOSITE state *SafeM_PowerVer* is entered and, within it, all behavior modeling activities related to the process of switching PDC on (power on process) is considered.

After the verification of power on, this second sequence of transitions {*ACT_HW-Reset[In (SafeM_PowerOnOk)], start60s, POSTOk, tsinc, VER_OP_MODE[In (NoPower)]*} allows the system to enter into the COMPOSITE state *SafeM_ResetVer* in order to verify whether the SWPDC performed correctly the tasks related to the reset process. A better approach is to separate these two test objectives into two distinct scenarios. SOLIMVA's normal scenario 50 (Fig. 15 and Fig. 16)

Table 18 A comparison between the expert and SOLIMVA approaches

Test Objectives	Expert	SOLIMVA
PDC initiation process	1	{OpMMgm, Init, -, -} = 43, 44, 45, 46, 47, 48, 49
Switching EPP Hxs on and off	2	{HwSwHnd, Safe, -, -} = 15, 16, 17, 18, 19, 20, 21
Changing software parameters in the Safety Operation Mode	3	{HwSwHnd, Safe, Hk, -} = 16
Processes of Power On and Reset	4	{HwSwHnd, Safe, -, PwrOn} = 16 {HwSwHnd, Safe, -, Reset} = 17, 18 {OpMMgm, Safe, -, PwrOn} = 50, 53, 54, 55, 56 {OpMMgm, Safe, -, Reset} = 52
Scientific Data Acquisition and Transmission in the Nominal Operation Mode	5	{DtAcqTx, Nom,Sci, -} = 71
Scientific Data Acquisition and Transmission in the Nominal Operation Mode, Robustness (command)	6	{DtAcqTx, Nom,Sci, -} = 71 {Inv, Nom,Sci, -} = 141
Houseekeeping Data Transmission in the Nominal Operation Mode	7	{DtAcqTx, Nom, Hk, -} = 72
Houseekeeping Data Transmission in the Nominal Operation Mode, Robustness (reception), Load new programs	8	{DtAcqTx, Nom, Hk, -} = 72 {PrLoad, Nom, Load, -} = 109 {Inv, Nom, Hk, -} = 142
Dump Data of Program Memory in the Nominal Operation Mode	9	{DtAcqTx, Nom, Dmp, -} = 73
Dump Data of Program Memory in the Nominal Operation Mode, Robustness (command and reception)	10	{DtAcqTx, Nom, Dmp, -} = 73 {Inv, Nom, Dmp, -} = 143
Dump Data of Data Memory (Pages 0 - 3) in the Nominal Operation Mode	11	bad strategy
Dump Data of Data Memory (Page 0) in the Nominal Operation Mode, Robustness (command and reception)	12	{DtAcqTx, Nom, Dmp, -} = 73 {Inv, Nom, Dmp, -} = 143
Dump Data of Data Memory (Page 1) in the Nominal Operation Mode, Robustness (command and reception)	13	{DtAcqTx, Nom, Dmp, -} = 73 {Inv, Nom, Dmp, -} = 143
Dump Data of Data Memory (Page 2) in the Nominal Operation Mode, Robustness (command and reception)	14	{DtAcqTx, Nom, Dmp, -} = 73 {Inv, Nom, Dmp, -} = 143
Dump Data of Data Memory (Page 3) in the Nominal Operation Mode, Robustness (command and reception)	15	{DtAcqTx, Nom, Dmp, -} = 73 {Inv, Nom, Dmp, -} = 143
Dump Data of Data Memory (Pages 4 - 7) in the Nominal Operation Mode	16	bad strategy
Dump Data of Data Memory (Page 4) in the Nominal Operation Mode, Robustness (command and reception)	17	{DtAcqTx, Nom, Dmp, -} = 73 {Inv, Nom, Dmp, -} = 143
Dump Data of Data Memory (Page 5) in the Nominal Operation Mode, Robustness (command and reception)	18	{DtAcqTx, Nom, Dmp, -} = 73 {Inv, Nom, Dmp, -} = 143
Dump Data of Data Memory (Page 6) in the Nominal Operation Mode, Robustness (command and reception)	19	{DtAcqTx, Nom, Dmp, -} = 73 {Inv, Nom, Dmp, -} = 143
Dump Data of Data Memory (Page 7) in the Nominal Operation Mode, Robustness (command and reception)	20	{DtAcqTx, Nom, Dmp, -} = 73 {Inv, Nom, Dmp, -} = 143

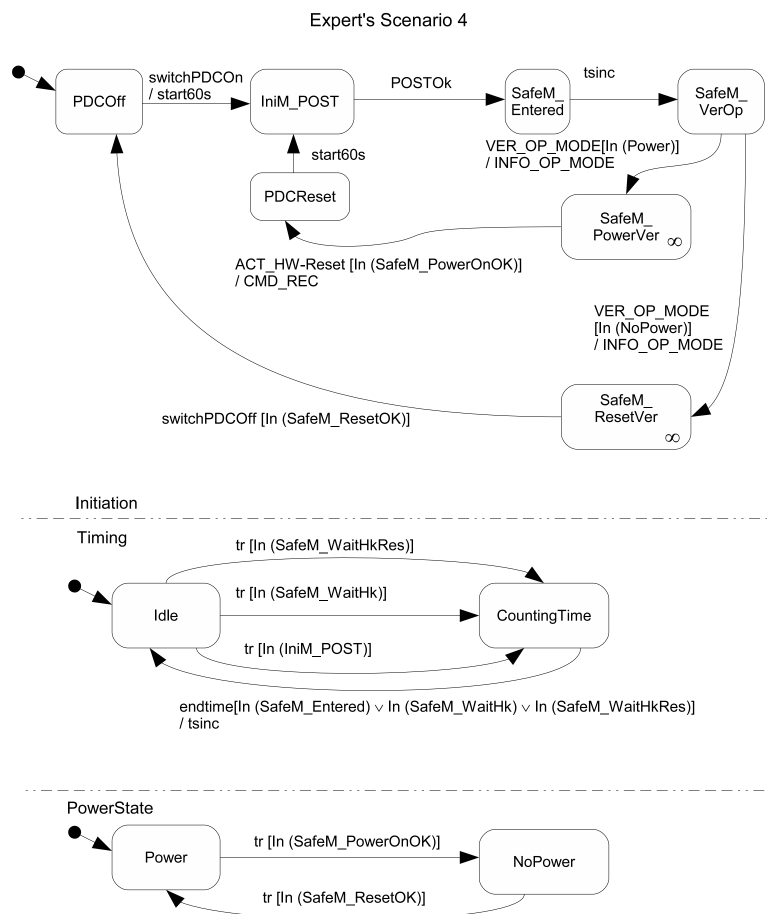


Fig. 14 Expert's scenario 4: main Statecharts model. Source: adapted from Santiago Júnior et al (2010)

addresses the power on process while SOLIMVA's normal scenario 17 covers the reset process (Fig. 17, Fig. 18, and Fig. 19).

The first difference among the models for expert's scenario 4 and those for SOLIMVA's scenarios 50 and 17 is the lack of parallelism in the models generated by SOLIMVA. Actually, all main Statecharts models of all expert's scenarios are AND states with parallel substates. We verified that presence of parallel states is not very relevant in our case because we are addressing to automatically create models for system and acceptance test case generation. In summary, we derive use case scenarios to stimulate the IUT one command at a time. Parallelism is not relevant in this context. Note that this is entirely different if we were to develop test cases based on models created by the development team in which parallelism is very likely to occur, and important to be accounted for. With this characteristic, models

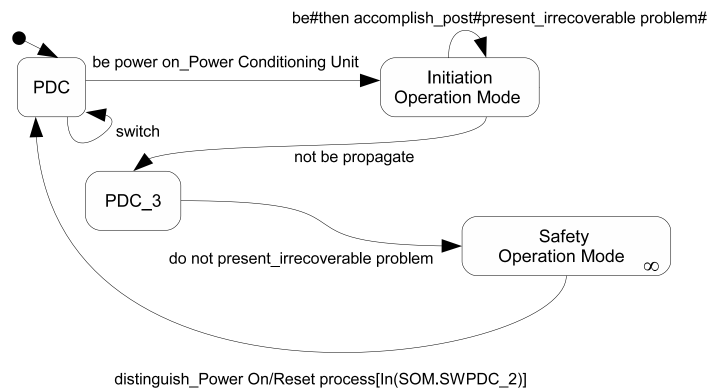


Fig. 15 SOLIMVA's normal scenario 50: main Statecharts model

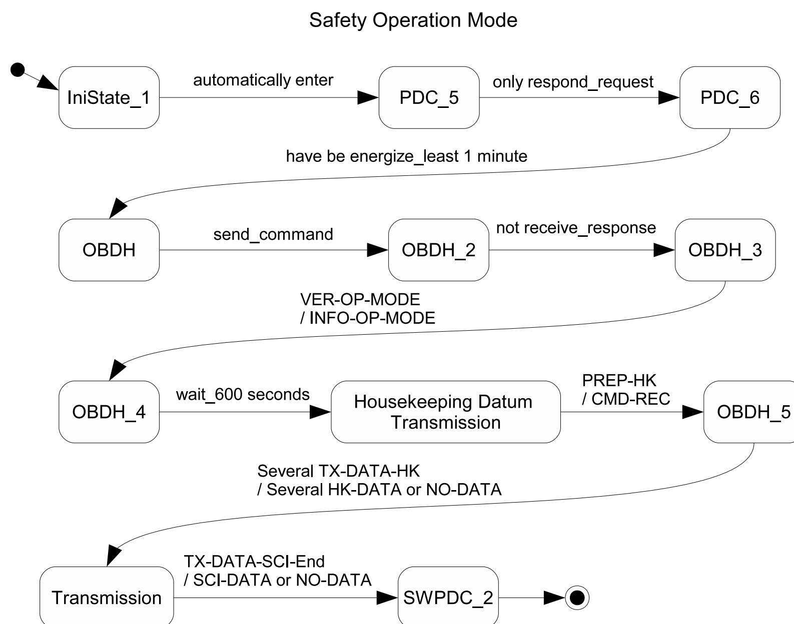


Fig. 16 SOLIMVA's normal scenario 50: *Safety Operation Mode*

generated by the SOLIMVA methodology are easier to read, and the translation from the Abstract Test Suite into the Executable Test Suite is not complex either. Besides, the expert's model missed

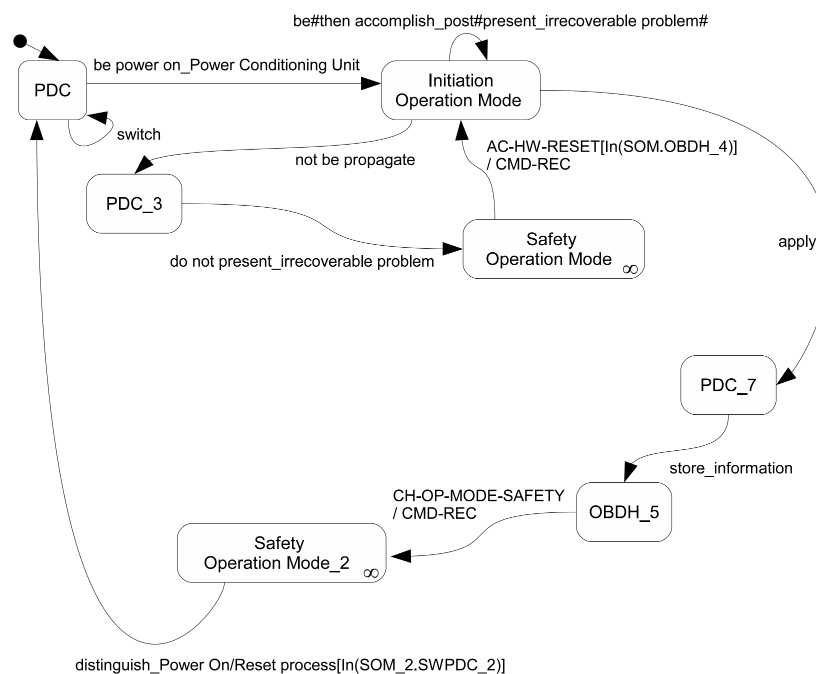


Fig. 17 SOLIMVA's normal scenario 17: main Statecharts model

one important transition that represents severe problems during the PDC initiation (initialization) process (the self transition in the *Initiation Operation Mode* state in Fig. 15 and in Fig. 17).

The separation of test objectives into two different scenarios provides a clear benefit in terms of strategy. Models generated for SOLIMVA's normal scenario 50, which have associated the power on verification, are very simple models. Checking whether the processes of power on and reset were successful is by means of Housekeeping data where SWPDC adds information to form a log. Hence, in the models for SOLIMVA's normal scenario 50, the *Safety Operation Mode* COMPOSITE state contains behavior to transmit Housekeeping data, i.e. the sequence of transitions {*wait_600 seconds, PREP-HK, Several TX-DATA-HK, TX-DATA-SCI-End*} in Fig. 16. However, for reset process addressed by SOLIMVA's normal scenario 17, the *Safety Operation Mode* COMPOSITE state has no such behavior because right after the initiation (initialization) process, the system shall be reset (*AC-HW-RESET[In(SOM.OBDH_4)]* in Fig.17) returning to the *Initiation Operation Mode* state. The behavior to transmit Housekeeping data appears in the *Safety Operation Mode_2* COMPOSITE state. Again, these differences favor simplicity. In order to contemplate these characteristics, the expert's models for scenario 4 were created with three hierarchy levels, say the main, the second and the third levels, adding complexity to understand such models.

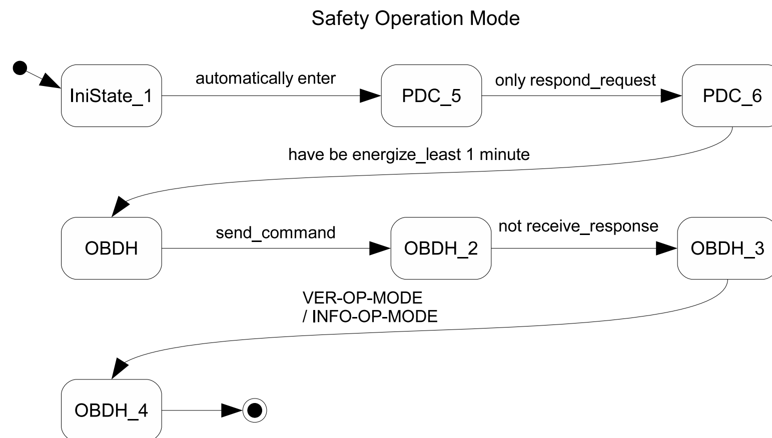


Fig. 18 SOLIMVA's normal scenario 17: *Safety Operation Mode*

Expert's scenario 8 is another issue. In this case, the test objectives aimed at two completely and unrelated goals such as Houseekeeping Data Transmission and loading new programs on the fly into PDC's Data Memory, and a third goal related to the robustness of SWPDC in situations where a command is not entirely received by PDC (reception). Loading new programs on the fly is a complex process in which the entire executable code is substituted for a new one during satellite operation. Hence, SOLIMVA will certainly drive the test designer to separate these three test objectives leading to a more coherent solution: normal scenario 72 "to transmit Housekeeping Data in the Nominal Operation Mode" (also foreseen in expert's scenario 7); normal scenario 142 "to address the problem of incomplete reception of commands"; and normal scenario 109 "to load new program into PDC's Data Memory on the fly in the Nominal Operation Mode". However, as mentioned earlier, normal scenario 109 needs to be unfolded and contributes with 6 unfolded scenarios replacing this normal scenario.

In Table 18, expert's scenarios 11 and 16 are "marked" as bad strategy. Indeed, the test objective related to expert's scenario 11 is covered by expert's scenarios 12 to 15. Similarly, the test objective associated to expert's scenario 16 is addressed by expert's scenarios 17 to 20. In case of SOLIMVA, the Dump Data service proposed in expert's scenarios 9 to 20 is covered by unfolding normal scenario 73. In order to illustrate that, let us consider expert's scenario 12 with the following test objectives: "Dump Data of Data Memory (Page 0) in the Nominal Operation Mode", "Robustness taking into account problems (inconsistent values) within commands sent to PDC (command)", and "Robustness addressing problem of incomplete reception of commands (reception)". The models related to expert's scenario 12 are shown in Fig. 20 and in Fig. 21.

In the COMPOSITE state *NomM_DmpPg0* (Fig. 21), the sequence of transitions {*P_DMP-DataP0-7FFFH-BFFFH, TX_DATA-Dmp*} relates to the test objective "Robustness taking into account problems (inconsistent values) within commands sent to PDC (command)". *P_DMP-DataP0-7FFFH-BFFFH* is an abbreviation for the following command defined in the PDC-OBDH

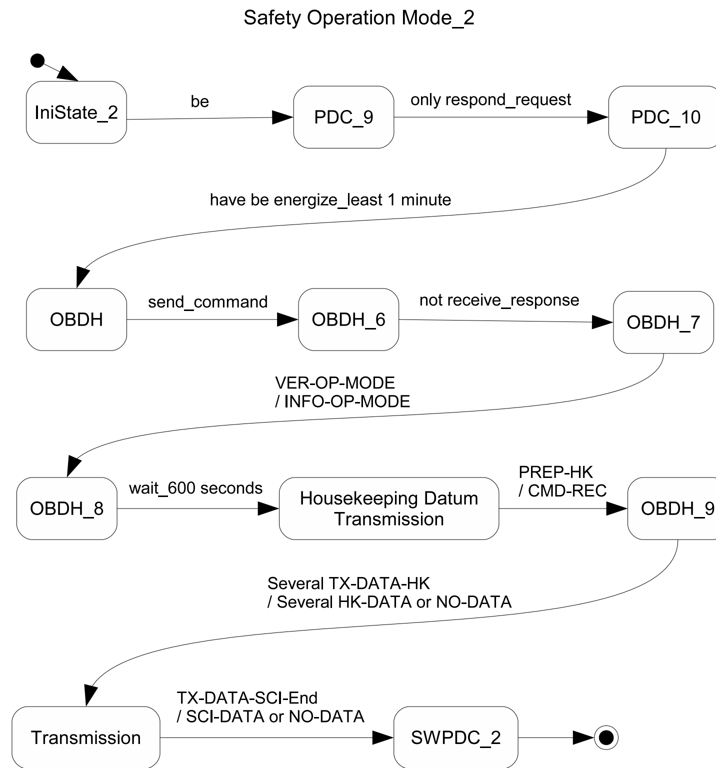


Fig. 19 SOLIMVA's normal scenario 17: *Safety Operation Mode_2*

Communication Protocol Specification: PREPARE DUMP FROM PAGE 0 OF DATA MEMORY, WITH INITIAL ADDRESS = 7FFFH AND FINAL ADDRESS = BFFFH. However, as shown in Fig. 10, the minimum memory address allowed for all pages of the Data Memory is 8000h. Thus, this command must not be processed by SWPDC due to the incorrect value of the initial address, i.e. lower than the minimum memory address allowed, and the expected result is a *timeout*. If a TRANSMIT DUMP DATA (*TX_DATA-Dmp*) is then sent, SWPDC must respond NO DATA (*NO_DATA*).

The sequence of transitions {*P_DMP-DataP0-8000H-BFFFH, TX_DATA-Dmp_14-1, TX_DATA-Dmp_0, TX_DATA-Sci*} relates to the test objective "Dump Data of Data Memory (Page 0) in the Nominal Operation Mode". Now, the initial address has an acceptable value (8000h is equal to the minimum memory address allowed) and data can be dumped normally from page 0. Then, SWPDC responds with Dump Data (*DMP_DATA-RSC14-1_1111* and *DMP_DATA-RSC0_830*).

The test objective "Robustness addressing problem of incomplete reception of commands (reception)" is modeled by the following sequence of transitions: {*P_DMP-DataP0-8000H-BFFFH, TX_DATA-Dmp_14-1, TX_DATA-Dmp_0, RET_ANSW, TX_DATA-Sci*}. Note the expected result

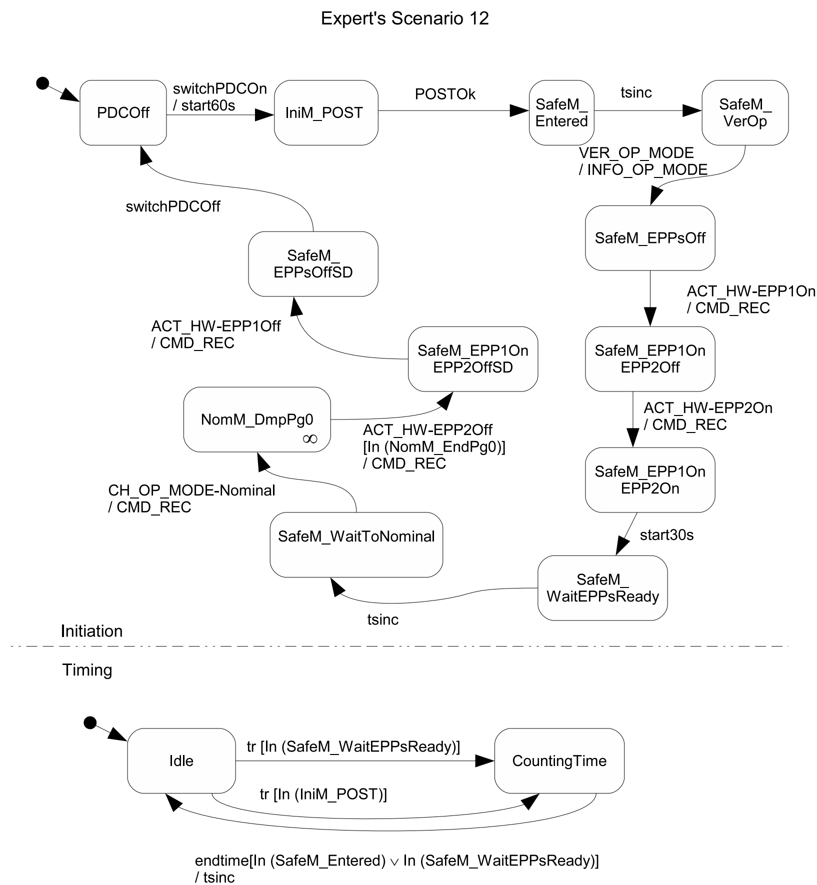


Fig. 20 Expert's scenario 12: main Statecharts model. Source: adapted from Santiago Júnior et al (2010)

timeout due to the test input datum TRANSMIT LAST DUMP DATA (*TX_DATA-Dmp_0*). The test designer assumed that a problem, probably in the physical transmission medium, occurred and only part of the command was received by SWPDC. The OBDH can then send a command to RETRANSMIT THE LAST DATA RESPONSE (*RET_ANSW*). Note that the same three test objectives apply to all other types of PDC's memory, i.e. Program Memory (expert's scenario 10) and the remaining pages of the Data Memory (expert's scenarios 13, 14, 15, 17, 18, 19, 20). The models for these other expert's scenarios are quite similar to the ones for expert's scenario 12.

Let us see how the SOLIMVA methodology covers expert's scenario 12. First, according to Table 13, unfolded scenario 73.2 (note DtP0 in every factor combination) is the one that relates to the Dump Data service from page 0 of the Data Memory. The set of factor combinations that is interpreted for generating unfolded scenario 73.2 is: {DtP0, InRng, Min}, {DtP0, Min, InRng},

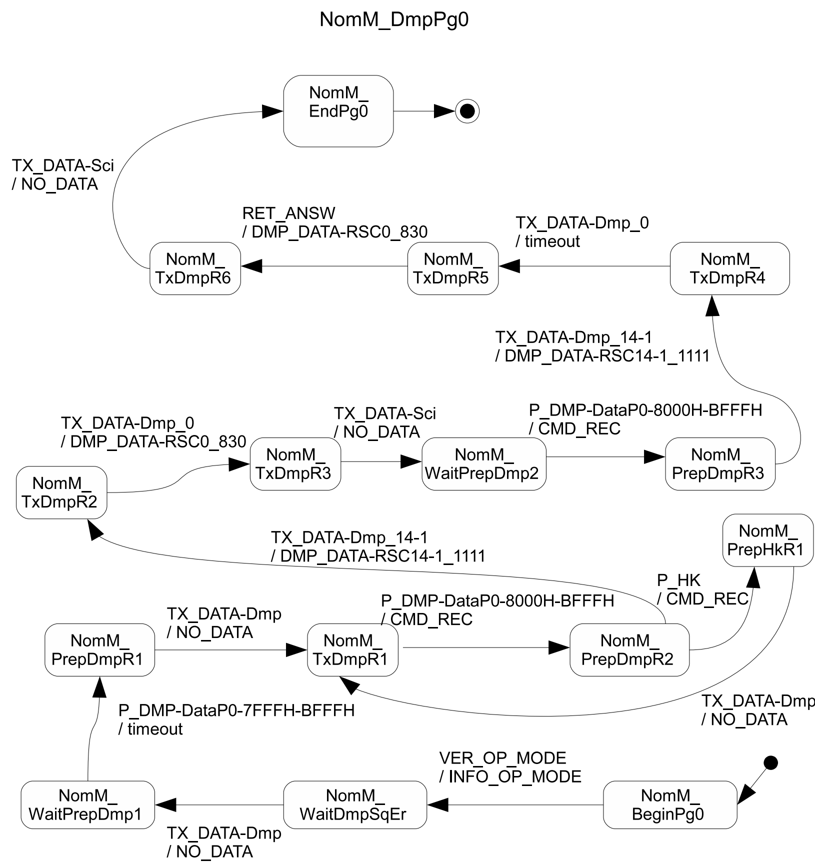


Fig. 21 Expert's scenario 12: *NomM_DmpPg0*. Source: adapted from Santiago Júnior et al (2010)

{DtP0, Max, LessMin}, {DtP0, LessMin, Max}, {DtP0, GreatMax, InRng}, {DtP0, Min, GreatMax}. All factor combinations except the second ({DtP0, InRng, Min}, {DtP0, Max, LessMin}, {DtP0, LessMin, Max}, {DtP0, GreatMax, InRng}, {DtP0, Min, GreatMax}) drive the test designer to add requirements in which inconsistent values of initial and/or final memory addresses are set. In other words, {DtP0, InRng, Min} implies that the initial memory address is In Range (between the Minimum and Maximum memory addresses allowed), and the final memory address is the Minimum memory address allowed. Thus, the initial address is greater than the final address and this is an incorrect setting. The conclusion is that these five factor combinations relate to the test objective "Robustness taking into account problems (inconsistent values) within commands sent to PDC (command)".

Nevertheless, the second factor combination ({DtP0, Min, InRng}) is consistent and it addresses the test objective "Dump Data of Data Memory (Page 0) in the Nominal Operation Mode". When translating the Abstract Test Suite into the Executable Test Suite, the test designer can substitute Min for 8000h (which is the only possible value for Min) and InRng for BFFFh, which is one of many possible values for InRng. Then, the same initial and final memory addresses proposed in the model for expert's scenario 12 can be selected.

The set of NL requirements that characterize unfolded scenario 73.2 is shown in Table 19. The combinatorial designs precisely direct the test designer to add requirements in accordance with the factor combinations. NL requirement SRS022 has the command PREPARE DUMP FROM PAGE 0 OF DATA MEMORY, WITH INITIAL ADDRESS = INRNG AND FINAL ADDRESS = MIN (*PREP-DMP-DTP0-INRNG-MIN*). This requirement was added due to factor combination {DtP0, InRng, Min}. NL requirements SRS023, SRS024, SRS025, SRS026, and SRS027 were added due to factor combinations {DtP0, Min, InRng}, {DtP0, Max, LessMin}, {DtP0, LessMin, Max}, {DtP0, GreatMax, InRng}, {DtP0, Min, GreatMax}, respectively.

Fig. 22 and Fig. 23 show the COMPOSITE state *Nominal Operation Mode* for SOLIMVA's unfolded scenario 73.2. The main Statecharts model is quite similar to the one generated for normal scenario 71 (Fig. 11) as well as are similar the COMPOSITE states *Safety Operation Mode* and *Safety Operation Mode_2*. The following sequences of transitions

- {*PREP-DMP-DTP0-INRNG-MIN, not stop_scientific datum acquisition, One TX-DATA-DMP, TX-DATA-SCI-End*},
- {*PREP-DMP-DTP0-MAX-LESSMIN, not stop_scientific datum acquisition, One TX-DATA-DMP, TX-DATA-SCI-End*},
- {*PREP-DMP-DTP0-LESSMIN-MAX, not stop_scientific datum acquisition, One TX-DATA-DMP, TX-DATA-SCI-End*},
- {*PREP-DMP-DTP0-GREATMAX-INRNG, not stop_scientific datum acquisition, One TX-DATA-DMP, TX-DATA-SCI-End*},
- {*PREP-DMP-DTP0-MIN-GREATMAX, not stop_scientific datum acquisition, One TX-DATA-DMP, TX-DATA-SCI-End*},

deal with situations where inconsistent values of initial and/or final memory addresses are accounted for. They were created as a result of the addition of the requirements due to all factor combinations except the second (Table 13). In these cases, all "expected results" related to commands PREPARE DUMP FROM PAGE 0 OF DATA MEMORY (*PREP-DMP-DTP0*) are *Timeout* precisely because of the wrong setting of addresses. These sequences of requirements address the test objective "Robustness taking into account problems (inconsistent values) within commands sent to PDC (command)".

On the other hand, the sequence of transitions {*PREP-DMP-DTP0-MIN-INRNG, stop_scientific datum acquisition, Several TX-DATA-DMP, TX-DATA-SCI-End*} is consistent and it addresses the test objective "Dump Data of Data Memory (Page 0) in the Nominal Operation Mode". The second factor combination was responsible for such sequence of transitions.

However, the test objective "Robustness addressing problem of incomplete reception of commands (reception)" is not covered not only by unfolded scenario 73.2 but also by any other unfolded scenario for other PDC's memories (73.1, 73.3, 73.4, etc.). In order to solve this problem, factor combination 143 ({Inv, Nom, Dmp, -}) can be used. Its interpretation defines normal scenario 143 and it deals with the incomplete reception of commands but considering all PDC's memories, i.e. the Program Memory and all pages of the Data Memory. This approach concentrates in a single scenario the

Table 19 Set of NL requirements that characterize unfolded scenario 73.2. Caption: Req = Requirement; Id = Identification

Req Id	**Req Description**
SRS001	The PDC shall be powered on by the Power Conditioning Unit.
SRS002	The PDC shall be in the Initiation Operation Mode after being powered on. The SWPDC shall then accomplish a POST. If PDC presents any irrecoverable problem, this computer shall remain in the Initiation Operation Mode and such a problem shall not be propagated to the OBDH.
SRS003	If PDC does not present any irrecoverable problem, after the initiation process, the PDC shall automatically enter into the Safety Operation Mode.
POCP001	The PDC can only respond to requests (commands) from OBDH after the PDC has been energized for at least 1 minute. If OBDH sends commands within less than 1 minute, the OBDH shall not receive any response from PDC.
RB001	The OBDH shall send VER-OP-MODE to PDC.
RB002	The PDC shall switch each Event Pre-Processor (EPP Hx, x = 1 or 2) on or off independently, when the OBDH sends distinct commands to perform such actions.
PECP001	Each EPP Hx can only respond to requests (commands) from PDC after each EPP Hx has been energized for at least 30 seconds. If PDC sends commands within less than 30 seconds to a certain EPP Hx, the PDC shall not receive any response from this EPP Hx.
RB003	The OBDH shall send CH-OP-MODE-NOMINAL to PDC.
RB001	The OBDH shall send VER-OP-MODE to PDC.
SRS022	Memory Dump data transmission shall start with PREP-DMP-DTP0-INRNG-MIN. In this case, the SWPDC shall not stop scientific data acquisition from EPP Hxs. After that, the OBDH can send One TX-DATA-DMP to PDC. The transmission shall be ended with TX-DATA-SCI-End.
SRS023	Memory Dump data transmission shall start with PREP-DMP-DTP0-MIN-INRNG. In this case, the SWPDC shall stop scientific data acquisition from EPP Hxs. After that, the OBDH can send Several TX-DATA-DMP to PDC. The transmission shall be ended with TX-DATA-SCI-End.
SRS024	Memory Dump data transmission shall start with PREP-DMP-DTP0-MAX-LESSMIN. In this case, the SWPDC shall not stop scientific data acquisition from EPP Hxs. After that, the OBDH can send One TX-DATA-DMP to PDC. The transmission shall be ended with TX-DATA-SCI-End.
SRS025	Memory Dump data transmission shall start with PREP-DMP-DTP0-LESSMIN-MAX. In this case, the SWPDC shall not stop scientific data acquisition from EPP Hxs. After that, the OBDH can send One TX-DATA-DMP to PDC. The transmission shall be ended with TX-DATA-SCI-End.
SRS026	Memory Dump data transmission shall start with PREP-DMP-DTP0-GREATMAX-INRNG. In this case, the SWPDC shall not stop scientific data acquisition from EPP Hxs. After that, the OBDH can send One TX-DATA-DMP to PDC. The transmission shall be ended with TX-DATA-SCI-End.
SRS027	Memory Dump data transmission shall start with PREP-DMP-DTP0-MIN-GREATMAX. In this case, the SWPDC shall not stop scientific data acquisition from EPP Hxs. After that, the OBDH can send One TX-DATA-DMP to PDC. The transmission shall be ended with TX-DATA-SCI-End.
RB004	The OBDH shall send CH-OP-MODE-SAFETY to PDC. After that, the PDC shall be in the Safety Operation Mode.
RB002	The PDC shall switch each Event Pre-Processor (EPP Hx, x = 1 or 2) on or off independently, when the OBDH sends distinct commands to perform such actions.
RB005	After switching both EPPHxs off via PDC, the OBDH shall switch the PDC off via the Power Conditioning Unit.

aforementioned test objective instead of modeling in every scenario for each PDC's memory as the expert proposed. A sample of the set of NL requirements that characterize normal scenario 143 is shown in Table 20.

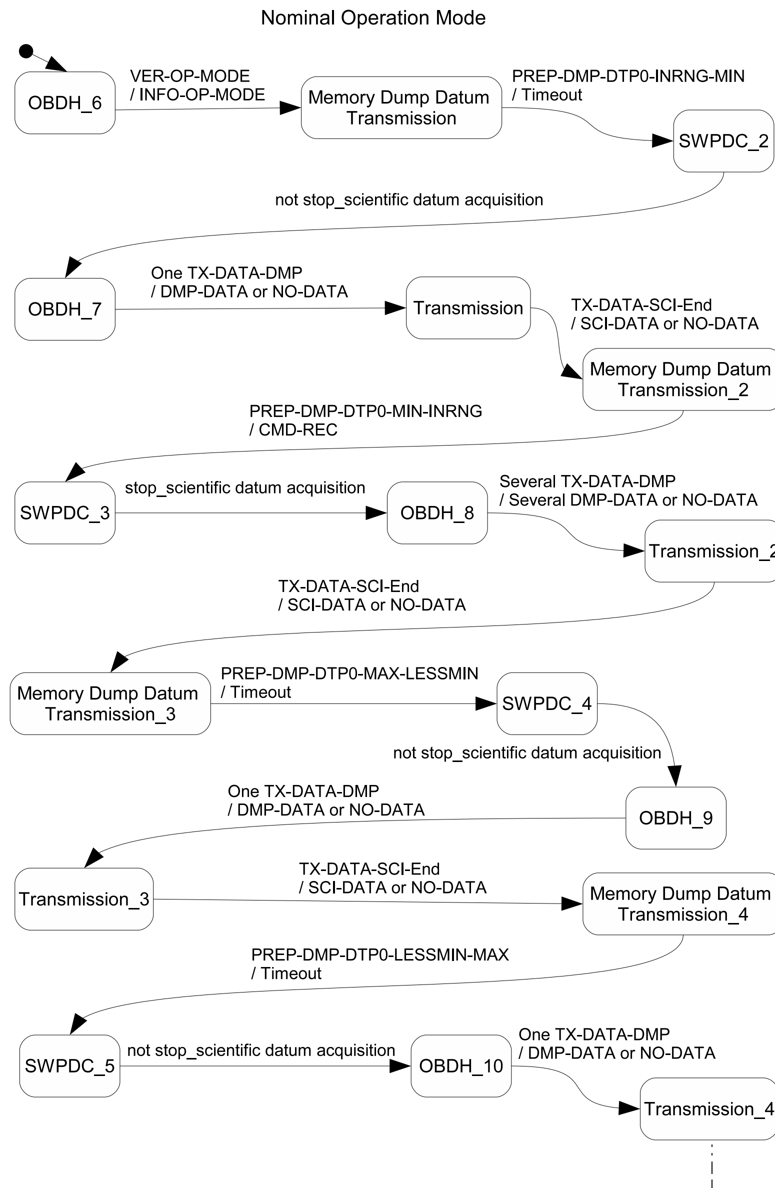


Fig. 22 SOLIMVA's unfolded scenario 73.2: *Nominal Operation Mode* (Part 1)

In Table 20, NL requirements SRS017, POCP021, SRS013 relate to the incomplete reception of commands when dumping data from the Program (PRG) Memory while SRS023, POCP021, SRS013

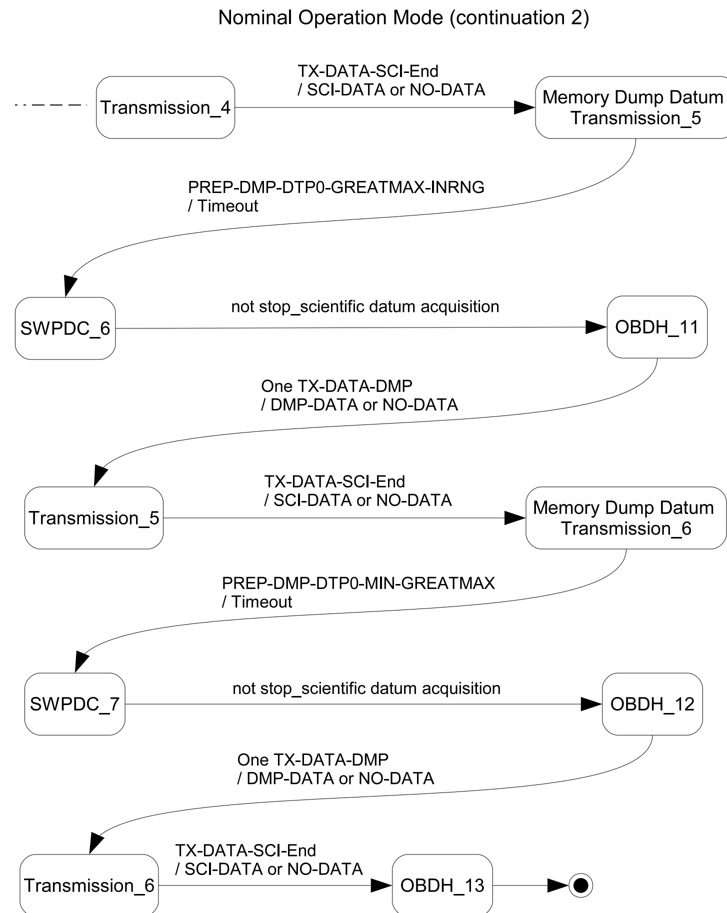


Fig. 23 SOLIMVA's unfolded scenario 73.2: *Nominal Operation Mode* (Part 2)

refer to page 0 of the Data Memory (DtP0). For the other pages of the Data Memory, it is enough to change the first of the three requirements and repeat the other two. Fig. 24 shows a piece of the COMPOSITE state *Nominal Operation Mode* for normal scenario 143 in which we can see the transitions that cover the test objective "Robustness addressing problem of incomplete reception of commands (reception)" when dealing with page 0 of the Data Memory (DtP0).

We have demonstrated how the SOLIMVA methodology adequately covered the test objectives associated to the expert's scenarios. As previously mentioned, the SOLIMVA methodology proposes a better strategy with test objectives clearly separated according to the directives of the combinatorial designs.

Table 20 Sample of the set of NL requirements that characterize normal scenario 143. Caption: Req = Requirement; Id = Identification

Req Id	**Req Description**
...	...
SRS017	Memory Dump data transmission shall start with PREP-DMP-PRG-INRNG-INRNG. In this case, the SWPDC shall stop scientific data acquisition from EPP Hxs. After that, the OBDH can send Several TX-DATA-DMP to PDC. The transmission shall be ended with TX-DATA-SCI-End.
POCP021	The PDC may not receive a command sent in its entirety. After identifying the beginning of a command frame, the PDC shall wait two times MAX-TRANSM-DELAY for the rest of the command. If this stipulated time expires, a timeout shall occur, the PDC shall abort the communication, the command shall be discarded, an event report shall be generated, and the PDC shall wait for a new OBDH's command.
SRS013	The SWPDC shall always maintain temporarily stored the last data response sent to the OBDH because the OBDH can demand the retransmission of this last data response.
SRS023	Memory Dump data transmission shall start with PREP-DMP-DTP0-MIN-INRNG. In this case, the SWPDC shall stop scientific data acquisition from EPP Hxs. After that, the OBDH can send Several TX-DATA-DMP to PDC. The transmission shall be ended with TX-DATA-SCI-End.
POCP021	The PDC may not receive a command sent in its entirety. After identifying the beginning of a command frame, the PDC shall wait two times MAX-TRANSM-DELAY for the rest of the command. If this stipulated time expires, a timeout shall occur, the PDC shall abort the communication, the command shall be discarded, an event report shall be generated, and the PDC shall wait for a new OBDH's command.
SRS013	The SWPDC shall always maintain temporarily stored the last data response sent to the OBDH because the OBDH can demand the retransmission of this last data response.
...	...

Our second goal was also achieved. We examined the Executable Test Cases generated based on the expert's and SOLIMVA's models. For instance, consider expert's scenario 5 whose main Statecharts model is shown in Fig. 25. As we have already discussed, this scenario is covered by SOLIMVA's normal scenario 71 whose models were previously shown (Fig. 11, Fig. 12, and Fig. 13). Choosing the all-transitions test criterion, there is only one Executable Test Case that composes the Executable Test Suite based on the expert's model:

{switchPDCOn/start60s, POSTOk/null, endtime/null, VER_OP_MODE/INFO_OP_MODE, ACT_HW-EPP1On/CMD_REC, ACT_HW-EPP2On/CMD_REC, start30s/null, endtime/null, CH_OP_MODE-Nominal/CMD_REC, VER_OP_MODE/INFO_OP_MODE, start10s/null, endtime/null, TX_DATA-Sci/SCI_DATA, start10s/null, endtime/null, TX_DATA-Sci/SCI_DATA, start10s/null, endtime/null, TX_DATA-Sci/SCI_DATA, start10s/null, endtime/null, TX_DATA-Sci/SCI_DATA, start10s/null, endtime/null, TX_DATA-Sci/SCI_DATA, start10s/null, endtime/null, TX_DATA-Sci/SCI_DATA, start10s/null, endtime/null, TX_DATA-Sci/SCI_DATA, start10s/null, endtime/null, TX_DATA-Sci/SCI_DATA, start10s/null, endtime/null, TX_DATA-Sci/SCI_DATA, start10s/null, endtime/null, TX_DATA-Sci/SCI_DATA, ACT_HW-EPP2Off/CMD_REC, ACT_HW-EPP1Off/CMD_REC, switchPDCOff/null}.

In order to compare the Executable Test Cases of both approaches, let us consider Table 21. Columns *Expert* and *SOLIMVA* show the test steps of the Executable Test Cases based on the expert's and SOLIMVA's strategies, respectively. Notice that all test steps of the single Executable Test Case of the expert's approach were satisfactorily covered by the three Executable Test Cases by employing SOLIMVA. Besides, SOLIMVA provides several benefits over the expert's approach.

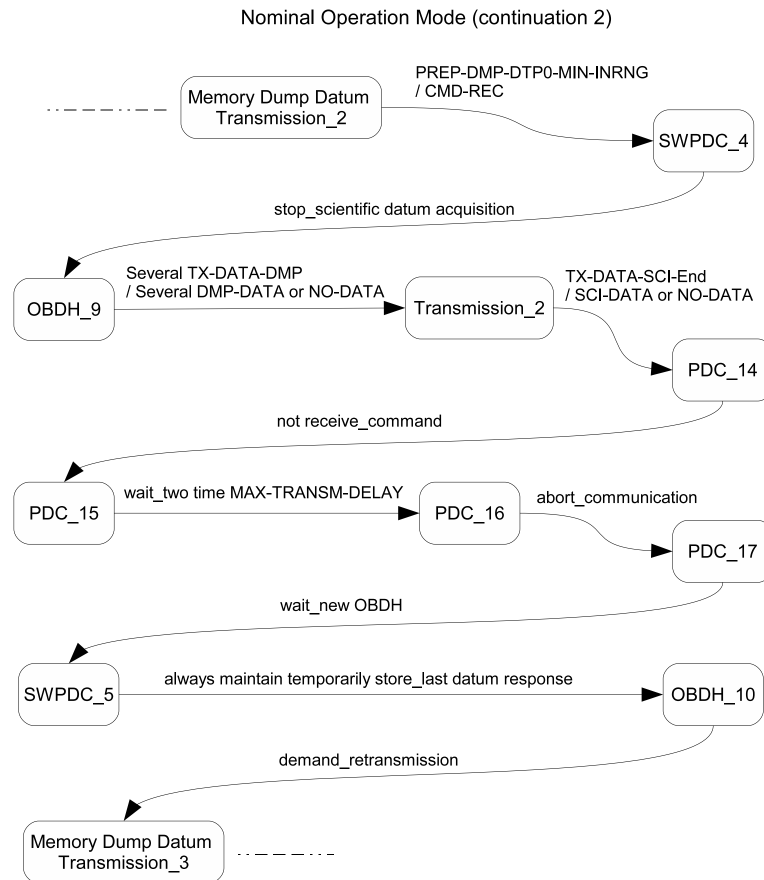


Fig. 24 SOLIMVA's normal scenario 143: a piece of the COMPOSITE state *Nominal Operation Mode*

First, the test steps {*start30s/null, endtime/null*} of the expert's approach were covered by a more detailed set of test steps. These test steps were based on NL requirement PECP001 (Table 14):

Each EPP Hx can only respond to requests (commands) from PDC after each EPP Hx has been energized for at least 30 seconds. If PDC sends commands within less than 30 seconds to a certain EPP Hx, the PDC shall not receive any response from this EPP Hx.

In case of the expert's approach, all the test designer should do is start a timer and wait 30 seconds. After this time interval, the PDC can begin to request data (Scientific, Test, Diagnosis) from EPP Hxs under OBDH request. However, the Abstract Test Cases derived by SOLIMVA contain the following test step from PECP001: {*only respond_request/null + have be energize_least*

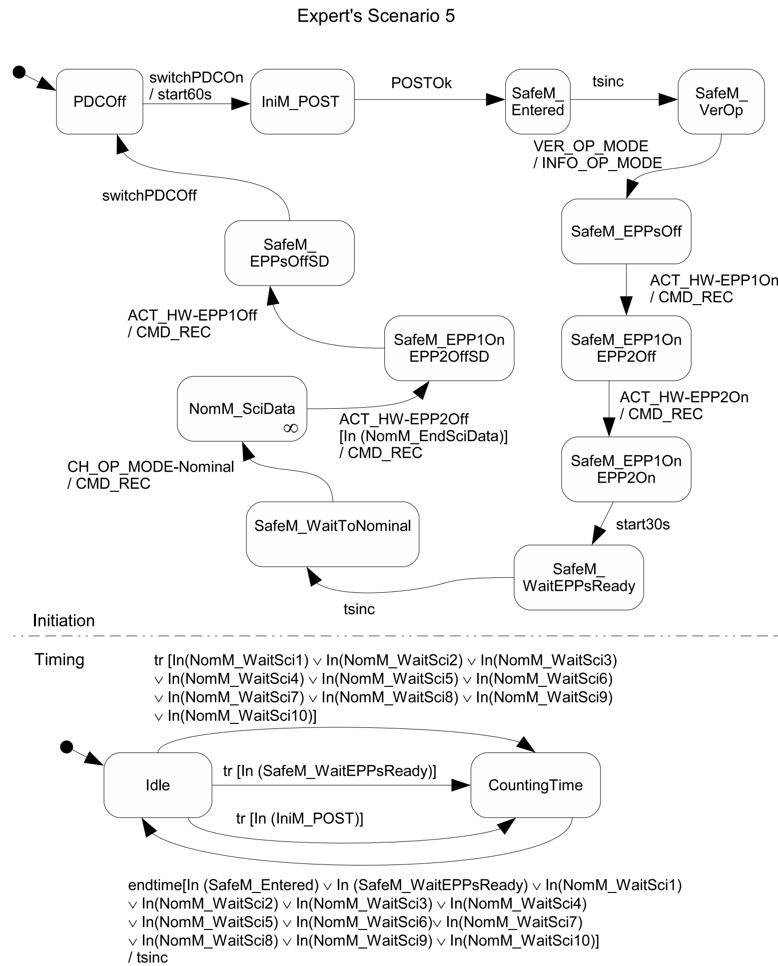


Fig. 25 Expert's scenario 5: main Statecharts model. Source: adapted from Santiago Júnior et al (2010)

30 seconds/null + send_command/null + not receive_response/null} (Table 16). When looking at these *input / output* pairs, the test designer realized that there are more relevant behaviors to be considered in the Executable Test Case, say the fact that if an EPP Hx receives commands within less than 30 seconds, the PDC shall not receive any response from this EPP Hx. Although this requirement is more related to EPP Hxs, it is important to verify whether PDC (SWPDC) acts adequately in such a situation. Then, instead of only waiting 30 seconds to start interacting with EPP Hxs, the test designer translated the abstract *input / output* pairs into a set of executable *test input data / expected result* aiming at the transmission of Test (TST) Data (one of the three types of data generated by EPP Hxs) within less than 30 seconds after each EPP Hx has been powered on

(*ACT-HW-EPPH1ON*, *ACT-HW-EPPH2ON*). As shown in Table 21, when the OBDH asks PDC to transmit Test Data (*TX-DATA-TST*) the expected result is no data (*NO-DATA*) because EPP Hxs are not yet available for communication. This improvement in the Executable Test Case due to the SOLIMVA methodology shows that looking at the Abstract Test Cases is more interesting rather than looking at a set of NL requirements in deliverables because the Abstract Test Cases provide a concise notation and emphasize the most relevant NL sentences, and some command/responses of Communication Protocol Specifications (in this particular case study) so that the test designer can derive more suitable Executable Test Cases.

Table 21 A comparison between expert's and SOLIMVA's Executable Test Suites. Caption: #ETC-E/-S = number of Executable Test Case within the Executable Test Suite derived by the expert (E) and SOLIMVA (S) approaches

#ETC-E	Expert	#ETC-S	SOLIMVA
1	switchPDCOn/start60s	1, 2	Action1, Action3
1	POSTOk/null	2	Action2
1	endtime/null	2	Action3
1	VER_OP_MODE/INFO_OP_MODE	2	VER-OP-MODE/INFO-OP-MODE
1	ACT_HW-EPP1On/CMD_REC	2	ACT-HW-EPPH1ON/CMD-REC
1	ACT_HW-EPP2On/CMD_REC	2	ACT-HW-EPPH2ON/CMD-REC
1	start30s/null, endtime/null	2	PREP-TST/CMD-REC, Action4, TX-DATA-TST/NO-DATA, TX-DATA-TST/NO-DATA, TX-DATA-SCI/NO-DATA, Action5
1	CH_OP_MODE-Nominal/CMD_REC	2	CH-OP-MODE-NOMINAL/CMD-REC
1	start10s/null, endtime/null, TX_DATA-Sci/SCI_DATA	2	Action4, TX-DATA-SCI/SCI-DATA
1	ACT_HW-EPP2Off/CMD_REC	2	ACT-HW-EPPH2OFF/CMD-REC
1	ACT_HW-EPP1Off/CMD_REC	2	ACT-HW-EPPH1OFF/CMD-REC
1	switchPDCOff/null	3	Action7
-	-	1	Action6
-	-	2	VER-OP-MODE/Timeout
-	-	2	Action5, PREP-HK/CMD-REC, TX-DATA-HK/HK-DATA, TX-DATA-SCI/NO-DATA
-	-	2	CH-OP-MODE-SAFETY/CMD-REC, VER-OP-MODE/INFO-OP-MODE

Although all test steps of the expert's Executable Test Case are addressed by SOLIMVA's Executable Test Suite, the opposite is not true. One very important test step missed in the expert's Executable Test Case is *Action6* which deals with severe problems during the PDC initiation (initialization) process. This test step relates to the translation of the self transition *be#then accomplish_post#present_irrecoverable problem#/null* in the *Initiation Operation Mode* state (Fig. 11, Fig. 15, and Fig. 17). As presented in Fig. 25, no self transition exists in the state that represents the Initiation Operation Mode (*IniM_POST*) in the expert's model. Then, we must not expect that any expert's Executable Test Case covers this behavior. The automated reading of NL requirements and the translation from the Abstract Test Suite into the Executable Test Suite make SOLIMVA an

interesting solution to minimize problems related to the incomplete creation of models for Model-Based Testing.

Another situation that is not present in the expert's Executable Test Suite is verifying the behavior of PDC if a command is sent by the OBDH within less than 1 minute since PDC has been energized (*VER-OP-MODE/Timeout*). This behavior relates to NL requirement POCP001 (Table 14). Furthermore, it is interesting to check some Housekeeping Data ({*Action5, PREP-HK/CMD-REC, TX-DATA-HK/HK-DATA, TX-DATA-SCI/NO-DATA*}) before entering into the Nominal Operation Mode in order to request Scientific Data. Finally, the expert's model did not consider the fact that in order to switch PDC off, it is recommended (there are requirements related to this recommendation) to put PDC in the Safety Operation Mode ({*CH-OP-MODE-SAFETY/CMD-REC, VER-OP-MODE/INFO-OP-MODE*}). Hence, states *SafeM_EPP1OnEPP2OffSD* and *SafeM_EPPsOffSD* in Fig. 25 are incorrectly named as being in the Safety Operation Mode (*SafeM*). Although this last issue might not be a huge problem for Statecharts model-based test case generation (more explanation about this in Section 6), again we show that the SOLIMVA methodology presents advantages over the expert's (manual) approach.

The conclusion is that the Executable Test Cases derived in accordance with the SOLIMVA methodology not only possessed similar characteristics with the expert's Executable Test Cases but also predicted behaviors that did not exist in the expert's strategy. Then, the models automatically created by the SOLIMVA methodology/tool are suitable for generating test cases. The suitability of the models are relevant because the test criteria used for test case generation based on FSMs and/or Statecharts are basically state-transition traversal rules. If the model is poorly developed, the Executable Test Cases might dictate a sequence of stimuli that are incoherent and in many situations impossible to be executed taking into account the real system.

6 Discussion

This section presents general remarks about the SOLIMVA methodology and its supporting tool. Each activity of the SOLIMVA methodology requires some sort of manual intervention but most of these activities have also a degree of automation. However, it is important to emphasize that in any initiative related to Software Testing Automation, human aspect is still very relevant. Even though there are lots of commercial and open source tools and frameworks available for this purpose, in almost all cases human interventions are still necessary to analyze test cases created by a tool, to evaluate coverage of test cases, to provide the expected result of the test cases, to verify whether a verdict was correctly asserted by an automated oracle, to generate models in a model-based approach, and so on. With respect to the activities of the SOLIMVA methodology (Fig. 3), the degree of manual intervention and automation is as follows:

1. Define and Input Dictionary, Update Dictionary. As we have previously pointed out, the test designer needs only to define and input, via SOLIMVA's Graphical User Interface, the Name (N) set, the Reactiveness (R) function, and the Hierarchy ($S_{TM}.Y$) function. The other sets of the Dictionary (Control, Self Transition) and any other auxiliary sets, for instance the sets that help in the generation of the BSAO tuples (Subsection 4.1), do not need to be changed and do not require manual intervention. Besides, the test designer does such a definition entirely in NL and thus no formalism is imposed to him/her;

2. Define Scenarios. The test designer must select factors and levels and interpret the factor combinations to derive scenarios. However, the factor combinations are automatically generated by an open source tool, TConfig;
3. Select and Input NL Requirements. This is done manually by the test designer. But, handling of NL requirements is automatically accomplished by the SOLIMVA tool;
4. Generate Model, Clear Requirements and Model. These two activities are completely and automatically performed by the SOLIMVA tool. The test designer needs only to start these activities via SOLIMVA's Graphical User Interface;
5. Generate Abstract Test Cases. This is automatically accomplished by the GTSC tool. The user needs only to start this activity via GTSC's Graphical User Interface;
6. Generate Executable Test Cases. This is done by the test designer.

SOLIMVA requires the test designer basically three types of efforts. First, it is the creation of a Dictionary which defines the application domain. However, from the point of view of test case generation addressing system and acceptance testing, in several situations the name of states in the Statecharts models are not very relevant. What matters is the *input event / output event* within the transitions in the Statecharts model that will be translated into *test input data / expected result* in the Executable Test Suite. Hence, the cardinality of the Name set does not need to be high. Moreover, as SOLIMVA uses the "divide and conquer" approach, i.e. splitting the interactions with the IUT into smaller scenarios, it is not mandatory to define an extensive Hierarchy function (ordered pairs) in order to predict different types of COMPOSITE states. Hence, cardinalities of the domain ($Y.I_P$) and codomain ($Y.O_P$) of $S_{TM}.Y$ do not need to be high either.

The Reactiveness function might or might not be huge. It depends on the way the deliverables are written. For instance, the SWPDC's Software Requirements Specification has several NL requirements where the commands defined in the PDC-OBDH Communication Protocol Specification are explicitly mentioned. Commands such as PREPARE HOUSEKEEPING DATA which we, for simplicity, abbreviated to PREP-HK, and TRANSMIT HOUSEKEEPING DATA abbreviated to TX-DATA-HK. In such situations, the test designer may prefer to add the ordered pair (*command, response*) in the Reactiveness function in order to make it easier the translation from the Abstract Test Suite into the Executable Test Suite. The explanation for this fact is that, since the ordered pair is in the Reactiveness function, it is very probable that they appear in the Statecharts (abstract) model and thus they will be part of the Abstract Test Cases. Since commands are executable then the translation from the abstract perspective into the executable one is simple.

However, if a requirements specification for space or other application domain has no explicit mention to low level commands or name of methods (functions) then the Reactiveness function may not have many ordered pairs. In this case, the input events of the transitions of the Statecharts model are pieces of NL sentences such as *only respond_request*, *have be energize_least 1 minute*, and the test designer shall translate them into executable *test input data / expected results*. Finally, the Dictionary is important because the lack of domain knowledge limits the applicability of systems based on unrestricted NL requirements (Ambriola and Gervasi, 2006).

The second type of effort is the need to translate Abstract Test Suites into Executable Test Suites. However, it is more interesting for the test designer to analyze a set of *input / output* pairs within the Abstract Test Cases and identify *test input data / expected results* of the Executable Test Suite rather than trying to find out what are the elements of a Executable Test Case directly from NL requirements documents. Besides, most of the translations accomplished for one scenario are reusable for other scenarios.

One may assert that the need to translate Abstract Test Suites into Executable Test Suites is a disadvantage of the SOLIMVA methodology. The point is that if the model is precise enough, some Model-Based Testing tools, including our GTSC tool, enable the generation of directly Executable Test Cases. Then, there would be no need to make such a translation. However, if we consider NL requirements deliverables like the ones in the SWPDC case study then it is important to realize that if a model has enough information so that a Model-Based Testing tool can generate Executable Test Cases it is because the test designer has translated from the NL requirements into the notation accepted by the tool before using the tool itself. To the best of our knowledge, there is no Model-Based Testing tool that can accept pure NL requirements documents and can, without any assistance from the user, generate directly Executable Test Suites. In other words, the translation from the abstract level into the executable level is manually done by the test designer before or after using the Model-Based Testing tool. In our case, we decided to make such a translation after using GTSC because we would like to use NL requirements as closely as possible in their original form due to the fact that this is a more realistic approach.

The last type of effort is the definition of scenarios by using combinatorial designs. This is without any doubt the most demanding effort. The test designer must not only define the factors and their respective levels but also to interpret the factor combinations to create the scenarios. He/she must decide whether a level must or must not be discarded in a certain factor combination and provide a set of scenarios that can cover most of the relevant interactions with the IUT. However, this is one of the main roles of a test designer within a Testing Team and such a professional must have a significant knowledge of the application domain to perform scenarios identification. But, knowledge of the application domain is mandatory: we can not envisage a test designer producing effective test cases with no or little expertise in the application domain. In short, SOLIMVA essentially requires the designer that he/she knows the application domain.

Another important observation refers to the strategy of separation of test objectives proposed by the SOLIMVA methodology as we have shown in Subsection 5.1. The question is whether the fact of disjoining behaviors, related to test objectives, as proposed by the methodology will not result in sets of Executable Test Cases that do not take into account each individual behavior. This particularly relates to Robustness testing. The approach of combinatorial designs coupled with the expertise of the test designer in the application domain can avoid this kind of situation. For example, recall that the three test objectives of the expert's scenario 12, two of them related to Robustness testing, were covered by SOLIMVA as follows: two of the test objectives were addressed by unfolded scenario 73.2, and one by normal scenario 143. As a matter of fact, for instance, there are 35 factor combinations derived by the combinatorial designs algorithm with *Inv* as the level for the command (Cmd) factor. Therefore, there are 35 normal scenarios to address problems like inconsistent values within the command frames, incomplete reception of commands or any other robustness feature the test designer might wish. These 35 scenarios spread over different PDC's operation modes (OpMode), Services, ways of initializing PDC (StartMode).

Although the SOLIMVA methodology proposes differently, the test designer may not want to separate test objectives and/or to neglect the interpretation of a level within a factor combination. Considering the SWPDC case study, the factor combination 170 is: {Inv, Inv, Hk, Reset}. One interpretation would generate normal scenario 170 as follows: "verification of behavior of SWPDC when receiving commands with inconsistent values or receiving commands incompletely, changing PDC's operation mode to an unspecified operation mode, generation, formatting, and transmission of Housekeeping Data, and verification of the reset process of PDC." The many unrelated test objectives denote a bad approach. But, the decision is up to the test designer.

Regarding scalability, the number of normal and unfolded scenarios derived by the test designer can be huge if too many factors and/or levels were selected. It is up to the test designer to choose the adequate number of factors and levels so that too many scenarios are not created. For instance, the level HwSwHnd is related to handling of hardware and software parameters. We could break this level into two, one for hardware parameters and other for software parameters. We decided to join these two features into one level precisely to decrease the number of scenarios.

Although the number of scenarios and consequently Statecharts models and Executable Test Suites can be huge, it is nowadays very common the use of frameworks and/or tools to automate the execution of test cases. We designed and implemented another tool to automate both system and acceptance test case execution and test process documentation generation (Santiago et al, 2008a). This tool has proven successful in shortening test case execution time in the context of Regression testing.

In order to generate model-based test cases using the SOLIMVA methodology for a different IUT in the space domain, other than the SWPDC software product, or even in other application domain, the user shall then repeat the activities proposed by the methodology taking into account the degree of manual intervention and automation as described earlier in this section. In other words, he/she shall define a new Dictionary (the Name (N) set, the Reactiveness (R) function, and the Hierarchy ($S_{TM}.Y$) function), define new scenarios (by choosing factors and levels, running the tool that has implemented the combinatorial designs algorithm, and interpreting the factor combinations to generate the scenarios), select and input a set of NL requirements which together characterize a single scenario, update the Dictionary (if needed), generate new Statecharts models (by using the SOLIMVA tool), generate Abstract Test Cases (by using the GTSC tool), translate the Abstract Test Cases into Executable Test Cases, and repeat this process, from the *Select and Input NL Requirements* activity, for all remaining scenarios.

Particularly for a different application within space domain, the amount of rework to apply SOLIMVA tends to be lower than in other application domains due to general similarities of the software products. For instance, the concept of *operation mode* is common to both the Space Segment and Ground Segment (ECSS, 2008) systems/subsystems. As we could see in the SWPDC case study, PDC's operation modes were elements of the Name (N) set, and they were also in the ordered pairs of the Hierarchy ($S_{TM}.Y$) function.

It is not in the scope of our work to conduct a study on cost and effectiveness of test suites generated by Model-Based Testing approaches. Particularly, the effectiveness of model-based test suites generated by Statecharts and FSMs is heavily dependent on two aspects. First, the model must be suitable enough in order to derive the generation of coherent Executable Test Suites. In the SWPDC case study, we showed that the models created by SOLIMVA were even superior in some respects than the models manually developed by an expert. The second point is the choice of the test criteria. This is an open issue in the academic community. Comparisons of fault detection effectiveness of some FSM (Sidhu and Leung, 1989; Souza, 2010) and Statecharts (Antoniol et al, 2002; Briand et al, 2004) test criteria have been published but there is no definite answer with respect to this issue. SOLIMVA does not address this second point.

Threats to external validity (Basili et al, 1996) exist with respect to the SOLIMVA methodology. Threats to external validity imply limitations to generalizing the results. We have applied the methodology to only one case study which is not enough. Another point is the fact that the methodology has been applied by only one of the authors of this paper, thus an expert in SOLIMVA.

The algorithm for automated detection of BSAO tuples (Fig. 7) is not general enough to work out with all NL sentences. However, it is important to stress that there are guidelines to elaborate

NL requirements in space application product development. For instance, the European Cooperation for Space Standardization (ECSS) has a standard to write Technical Requirements Specifications (ECSS, 2009). Among other features, this standard provides "recommendations for the wording of requirements" stating how to write requirements, how modal verbs should be used, and terms to be avoided within the NL requirements. Directives like this and our experience in the space application domain drove the design and implementation of the BSAO tuples generation algorithm.

Our approach is limited by the external tools that we used. For instance, as any other POS tagger tool, the *Stanford POS tagger* is not 100% free of failures. We rely on its output to identify the BSAO tuples. Let us consider the following requirement:

The SWPDC shall format scientific data from each EPP Hx (x = 1 or 2).

The *Stanford POS tagger* recognizes "format" as a *common noun, singular or mass* (POS tag "NN") instead of a *verb, base form* (POS tag "VB"). This is a problem that we must not try to solve. It is not in the scope of our work to develop a new POS tagging algorithm and tool to identify lexical categories of words: there are many of them available, and our idea was to use a publicly available POS tagger (Stanford) to help with our main goal which is to generate the Statecharts models from NL aiming at system and acceptance model-based test case generation.

The following advantages should be emphasized if a test designer decides to apply the SOLIMVA methodology and its supporting tool rather than relying on a completely manual ad hoc strategy like the expert's approach discussed in Subsection 5.1:

1. Compared to other formal methods, FSMs and Statecharts are relatively easy to understand. The SOLIMVA methodology aims precisely to avoid that a test designer, who is not experienced even with these simple modeling techniques, needs to develop the models from scratch. We have observed that professionals from aerospace application domain and graduate students find it difficult in translating pure NL Software Requirements Specifications into Statecharts or FSMs models in order to address system and acceptance testing. SOLIMVA requires the user to define the application domain, by means of a Dictionary, and scenarios. However, the methodology and its supporting tool provide a "first" model so that the test designer can start working. If the test designer does not feel comfortable with the model, he/she can try to improve it (*manual refinement*);
2. We have previously stated that the translation from the abstract level into the executable level is somehow manually accomplished by the test designer before or after using a Model-Based Testing tool. SOLIMVA proposes to make such a translation after using the tool (GTSC) in order to use NL requirements as closely as possible in their original form due to the fact that this is a more realistic approach. In addition, verifying a set of *input / output* pairs within the Abstract Test Cases rather than directly looking at NL requirements documents is a more feasible way to perform the translation from the Abstract Test Suite into the Executable Test Suite. Moreover, the Abstract Test Cases provide a concise notation and emphasize the most relevant NL sentences, allowing the test designer to generate more suitable Executable Test Suites;
3. The SOLIMVA methodology and its supporting tool allow to automatically start reading NL requirements. This fact added to the translation from the Abstract Test Suite into the Executable Test Suite make our methodology an interesting solution to minimize problems related to the incomplete/inconsistent creation of models for Model-Based Testing;
4. SOLIMVA suggests a precise, systematic and mathematical-based solution to identify scenarios for system and acceptance test case generation by means of combinatorial designs;

5. The separation of test objectives proposed by SOLIMVA results in a set of scenarios with goals more closely related and therefore a better strategy is achieved.

7 Related work

This section will be divided into three subsections as follows.

7.1 Model-Based Testing

This subsection will provide an overview of works related to Model-Based Testing. We follow the definition of Model-Based Testing given in Section 1.

Briand and Labiche (2002) presented the *Testing Object-orienTed systEms with the unified Modeling language* (TOTEM) approach based on UML diagrams addressing functional system testing. Test requirements are derived from use case diagrams, use case descriptions, interaction diagrams (sequence or collaboration) associated with each use case, and class diagrams (composed of application domain classes and their contracts).

FSMs (Lee and Yannakakis, 1996) and Statecharts (Harel, 1987; Harel et al, 1987) are a few examples of modeling techniques commonly used for testing. Simplicity is one of the key advantages in using FSM and this technique has been in use for modeling reactive systems and protocol implementations for a long time. Once an IUT is modeled as a state-transition diagram representing an FSM, several test criteria like Transition Tour (TT), DS, UIO (Sidhu and Leung, 1989), W (Chow, 1978), switch cover (1-switch) (Pimont and Rault, 1976), and state counting (Petrenko and Yevtushenko, 2005) can be used to generate test cases.

A Model-based approach to generate a set of conformance test cases for interactive systems was proposed by Paradkar (2003). The approach presents extensions to both the Category-Partition method (Ostrand and Balcer, 1988) and the *Test Specification Language* (TSL) (Balcer et al, 1989). Test case generation is based on the extraction of a Finite State Automaton (FSA) from a specification written in an extended version of TSL, known as *Specification and Abstraction Language for Testing* (SALT).

An algorithm that generates a partition of the input domain from a Z specification was introduced by Hierons (1997). This partition can be used both for test case generation and for the production of an FSA. This FSA can then be used to control the testing process. This method generates a large FSA making this approach difficult for test case generation addressing large software systems (Paradkar, 2003).

Conformance and Fault Injection (CoFI) is another model-based test case generation methodology (Ambrósio et al, 2007). In CoFI, the system behavior is partially represented in state-transition diagrams representing FSMs, and test cases are generated based on such FSMs. The methodology requires the test designer first the identification of a set of scenarios (services) to stimulate the IUT, and then the precise definition of the IUT's interfaces. CoFI is basically a use case-based testing approach (Frohlich and Link, 2000; Bertolino and Gnesi, 2003) with some emphasis in hardware fault tolerance. One limitation of CoFI is not to provide precise guidelines for the test designer to identify the usage scenarios (services): the test designer must do it on an ad hoc basis.

Several approaches have been proposed to generate test cases based on Statecharts. Binder (1999) adapted the W test criterion to a UML context and named it round-trip path testing, in which

flattening a Statecharts model is a prerequisite before using the criterion itself. Santiago et al (2006) proposed a methodology to transform hierarchical and concurrent Statecharts into FSMs in order to generate test cases, with the support of the PerformCharts tool (Vijaykumar et al, 2006).

The *Statechart Coverage Criteria Family* (SCCF) was proposed by Souza (2000). It is a family of testing coverage criteria for Statecharts models. Test requirements established by the SCCF criteria are obtained from the Statecharts reachability tree (Masiero et al, 1994). Antoniol et al (2002) presented a study whose main goal was to analyze cost and efficiency of the Binder's round-trip path criterion. Briand et al (2004) showed a simulation and a procedure to analyze cost and efficiency among three test criteria proposed by Offutt and Abdurazik (1999) and the very same round-trip path.

A system testing approach to coverage of *elementary transition paths* was proposed by Sarma and Mall (2009). The technique relies on the derivation of a System State Graph (SSG) based on UML 2.0 use case, sequence, and Statecharts models. The test criterion which their method aims to satisfy is *transition path coverage* which states that each elementary transition path p of the SSG must be covered at least once by a test suite T. Sarma-Mall's work presents some limitations but the most severe of all relates to the the fact that a loop is either not executed at all or it is executed only once. Thus, the authors did not address one of the major problems in path testing: in general, a program containing loops will have an infinite or undetermined number of paths (Howden, 1976).

As mentioned in Section 3, GTSC is an environment 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, 2008b). We have been developing and using it in the context of research projects. Recently, we have been working in a proposal for combining Statechart-based and Z-based testing (Cristiá et al, 2010). GTSC has been used in conjunction with the Fastest tool (Cristiá and Monetti, 2009) in order to meet this goal.

Hierons et al (2009) published a survey which described formal methods, software testing, and a number of ways in which a formal specification can be used in order to assist testing. They divided formal specification languages into several categories, among others *Model-Based Languages* (e.g. Z(Spivey, 1989)), *Finite State-Based Languages* (e.g. FSM[7](Lee and Yannakakis, 1996), Statecharts (Harel, 1987)), and *Process Algebra State-Based Languages* (e.g. Communicating Sequential Processes (CSP) (Hoare, 1985)). They concluded that "software testing benefits from the presence of a formal specification in a number of important ways. In particular, the presence of a formal specification aids test automation and allows the tester to reason about test effectiveness".

The SOLIMVA methodology relies on the GTSC environment in order to generate test cases. However, an important feature of our methodology is to provide a formal manner, by means of combinatorial designs, to identify scenarios for system and acceptance test case generation.

7.2 Software testing based on NL requirements

We found only one publication in the literature that proposed to generate test cases starting from NL requirements: *Text Analyzer* (Sneed, 2007). It is a tool that supports black box testing and it is intended to be used for system and acceptance testing. *Text Analyzer* needs heavy intervention from the user to define the application domain. The tool first scans the text in order to identify all nouns. These nouns are displayed to the test designer who decides which ones are considered

[7] Traditionally, an FSM is seen as a computational model and not as a language (Hopcroft and Ullman, 1979).

pertinent objects of the IUT. Such objects are in turn the elements the test cases relate to. This task can be very time-consuming depending on the complexity of the requirements specification. The user must also identify *keywords* used in the requirements text (e.g. *INPT = this word indicates a system input*). This is another activity that seems to require considerable time and that can make the approach less attractive, especially considering complex NL requirements documents. Moreover, *Text Analyzer* does not make use of formal methods (languages, models) and their benefits with respect to test case generation like the SOLIMVA methodology proposes.

7.3 Translation of requirements

This subsection presents some works related to the translation of requirements from one notation into a different notation. In particular, approaches that translate NL requirements into formal methods can be quite convenient because they relieve professionals of the cost of learning a formal method but, at the same time, provide the requirements converted so that tools may be used aiming at Model-Based Testing and Formal Verification.

CIRCE is an environment that supports modeling and analysis of requirements described in NL (Ambriola and Gervasi, 2006, 1997). The tool parses and transforms NL requirements into a forest of parse trees. To do that, CIRCE uses a domain-based parser called CICO. By defining requirements in accordance with the formal model embedded in CIRCE, the tool can generate models like state-transition diagrams allowing the user to analyze problems in requirements.

CIRCE seems to be a remarkable tool. However, the greatest issue related to CIRCE is the easiness to express the domain. In other words, application domain must be expressed by a user by means of designations and definitions which, in turn, must be written using a formal syntax. A Requirements Engineer must declare designations using lots of tags and he/she must perform a deep analysis of the NL requirements to accomplish that. The need to write *Model, Action, Substitution* (MAS) rules, which are formal rules that drive the CICO's parsing algorithm, can be a significant obstacle for practitioners who are not specialized in the tool and wish to use it.

The *Natural Language - Object Oriented Production System* (NL-OOPS) is a tool that supports analysis of unrestricted NL requirements by extracting the classes and their associations for use in creating class models (Mich, 1996). The unrestricted NL analysis is obtained using as a core the NL processing system *Large-scale, Object-based, Linguistic Interactor, Translator, and Analyser* (LOLITA) (Morgan et al, 1995). However, the lack of domain knowledge is a limiting factor for using systems based on unrestricted NL requirements taking into account real and complex projects (Ambriola and Gervasi, 2006).

Kim and Sheldon (2004) presented a method that models and evaluates NL software requirements specifications using the Z formal language and Statecharts. Their method transforms a NL specification into a Z specification which in turn derives the Statecharts models (actually, State/Activity charts). The goal was to analyze the integrity of a specification in terms of completeness, consistency, and fault-tolerance. Their work presented some interesting results but the transformations proposed were heavily dependent on human skill, and there is no evidence that a tool was developed to automate the defects detection.

Gervasi and Zowghi (2005) proposed a formal framework for identifying, analyzing, and managing inconsistency in NL requirements derived from multiple stakeholders. A prototype tool, CARL, was developed and they focused on a particular kind of inconsistency, *logical contradiction.* The authors

claim that the framework supports the detection of both explicit and hidden inconsistencies[8]. For dealing formally with inconsistency, first requirements expressed in controlled NL are automatically parsed and translated into propositional logic formulae. Once the specification is represented as sets of propositional logic formulae, a theorem prover and a "model checker"[9] are used aiming at detecting inconsistencies. Despite these remarkable features, and as well as CIRCE (Ambriola and Gervasi, 2006), CARL suffers from the same problem regarding the likely need to write new MAS rules depending on the domain.

Attempto Controlled English (ACE) is a controlled NL specifically constructed to write specifications (Fuchs et al, 1999, 2000). ACE is a subset of standard English, and its specifications are computer-processable and can be translated into first order logic. Although it is claimed that ACE combines NL with formal methods, and it is easier to learn and use than formal languages, the language is very restricted. Very restricted versions of NL are often comparable to formal languages with NL-like keywords (Ambriola and Gervasi, 2006). This fact may limit the applicability of ACE in real projects.

Liang and Palmer (1994) discussed the correspondence between NL requirements sentence structure patterns and events/transitions concepts in state-transition diagrams representing Extended FSMs. The goal was how to extract events and transitions from conditional sentences. In order to support such extraction, a pattern matching and clustering-based approach was proposed. There was no tool to automate this approach. The identification of events/conditions/actions per se is accomplished manually by the user by examining clusters.

Fraser et al (1991) proposed to bridge the gap between informal and formal requirements specification languages. They used Structured Analysis (SA), by means of Data Flow Diagrams (DFDs), and the Vienna Development Model (VDM) as surrogates for informal and formal languages, respectively. This proposal did not start the translation directly from NL requirements.

Lu et al (2008) presented the *Model-driven Object-oriented Requirement Editor* (MOR Editor), a tool that supports requirement document modeling and model-driven document editing. It is possible to transform the informal (NL) requirements into *Model-based OO Requirement Models* (MOORMs), templates from which software requirements can be instantiated. Their approach lacks mathematical formalism and the translation from NL requirements into MOORMs is not straightforward.

Java Requirement Analyzer (J-RAn) is a tool that implements a Content Analysis technique to support the analysis of inconsistency and incompleteness in NL requirements specifications (Fantechi and Spinicci, 2005). Based on the NL document, this technique explores the extraction of the interactions between the entities described in the specification as Subject-Action-Object (SAO) triads. These SAO triads are obtained with the help of the *Link Grammar Parser* (Sleator and Temperley, 1993), a syntactic parser of English based on *Link Grammar*, a formal grammatical system. J-RAn was applied in a very simplified case study and, even so, a significant number of SAO triads were incorrectly extracted (21%) as a large number of extractions were not detected as well (16%).

Comparing the SOLIMVA methodology and its supporting tool with the works presented in this and in previous subsections, we may stress the following advantages:

1. Easiness of use. As mentioned in Section 3, SOLIMVA does not require the user any knowledge in formal methods and their respective notations to define the application domain, like CIRCE

[8] In these cases, the inconsistency occurs due to the consequences of some requirements rather than the requirements themselves.

[9] Actually, CARL does not really apply model checking according to its most common definition (Clarke and Lerda, 2007; Baier and Katoen, 2008).

(Ambriola and Gervasi, 2006) and CARL (Gervasi and Zowghi, 2005) do. Besides, the definition of the application domain as proposed in SOLIMVA seems to be far more simpler than the one presented in *Text Analyzer* (Sneed, 2007);
2. Writing of NL requirements. At first, the SOLIMVA tool does not impose any constraint in writing NL requirements. However, we can not say that the tool follows the unrestricted NL approach, like NL-OOPS (Mich, 1996), because it is necessary to define the application domain by means of a Dictionary. Hence, the limitations of the unrestricted approach are not an issue with SOLIMVA. On the other hand, we can not also say that the SOLIMVA tool is based on a controlled NL approach like CIRCE, CARL, and the very restricted controlled NL ACE (Fuchs et al, 1999, 2000). This implies that the user has more freedom to use our tool;
3. Identification of scenarios. SOLIMVA proposes a formal manner, by means of combinatorial designs, to identify scenarios for system and acceptance test case generation. Other methodologies, like CoFI (Ambrósio et al, 2007), adopt an ad hoc approach in order to achieve such a purpose, and this fact may limit the strength of such proposals;
4. Semantics. From the perspective of model generation based on NL sentences, some tools require the user to manually provide explicit definitions for concepts. For instance, in CIRCE, the user provides such explicit definitions by means of *definitions*, an element of the *requirements document model* of the tool. In this case, we can say that the semantics of the model is somehow manually provided by the user. A similar observation can be made with respect to *Text Analyzer* when the user must identify keywords in the text. In SOLIMVA, we tried to automate the semantics related to the generated model by adapting a word sense disambiguation algorithm in order to identify self transitions in the resulting Statecharts model;
5. Automation. Although many publications presented above support the automated translation from NL requirements into another notation, others do not have such a characteristic, for instance Kim and Sheldon (2004) and Liang and Palmer (1994) proposals. This is another important feature of the SOLIMVA tool;
6. Starting from NL requirements. Some publications, like the one of Fraser et al (1991), aim to bridge the gap between informal and formal requirements specification languages. However, they do not begin to approach directly from NL requirements as SOLIMVA does;
7. Mathematical formalism. In order to generate the test cases, the SOLIMVA methodology and its supporting tool translate the NL requirements into a formal method, the Statecharts language. The MOR Editor (Lu et al, 2008) lacks mathematical formalism.

8 Conclusions

The greatest motivation of this work is to bridge the gap between the state of the art (formal methods, computational linguistics techniques, combinatorial designs) and the state of the practice (NL requirements deliverables). We believe this is an important approach towards a wider use of the theory proposed by the academic community in real projects in the industry, and in institutes of research and development.

This paper presented the SOLIMVA methodology which aims to help test designers to generate test cases based on behavioral models taking into account embedded reactive systems. The methodology assumes that the deliverables (artifacts) constituting the basis for system and acceptance test case generation, such as software requirements specifications, are mostly developed in NL. The generation of test cases in an automated fashion directly from NL documents, as proposed in this

work, is a challenge since NL presents serious shortcomings like ambiguity, poor understandability, incompleteness, and inconsistency.

Summarizing what we have mentioned earlier, the SOLIMVA methodology has some benefits over other research such as: (i) it provides a formal manner, by means of combinatorial designs, to identify scenarios for model-based system and acceptance test case generation; (ii) it is supported by a formal method but it does not require the user any skills related to formal notations; (iii) it does not present some problems of other unrestricted NL approaches, such as limitation of applicability, or controlled NL approaches, such as the necessity of strict compliance with predefined standards of writing; (iv) it has a solution to automate the identification of self transitions in the resulting Statecharts model by adapting a word sense disambiguation algorithm; (v) it begins to generate the models and hence the Executable Test Suites directly from NL requirements. All these points show that our methodology can give an important contribution to the software Verification and Validation process.

However, the research presented in this work should evolve so that the SOLIMVA methodology can be applied on a broader scale. We need to investigate the feasibility of the SOLIMVA methodology in other case studies not only in the space application domain but also in other embedded reactive system domains. Besides, we intend to elaborate an empirical study with other professionals and analyze the impact of the introduction of SOLIMVA in other settings. The algorithm for identification of BSAO tuples should be improved, and we will develop a mechanism to automatically translate Abstract Test Cases into Executable Test Cases. Furthermore, we will implement a combinatorial designs algorithm, such as IPOG (Lei et al, 2007), within the SOLIMVA tool to eliminate the need to use an external tool (TConfig) for generating Mixed-Level Covering Arrays.

Despite the explanations given in Section 6, scalability is still a major issue. The SOLIMVA methodology is appropriate up to medium sized space application software products. Our approach needs to be improved in order to deal with large software systems. One way towards this improvement is to optimize (reduce) the number of generated scenarios by using techniques to merge scenarios or even to discard some of them.

Other important future work is the detection of defects such as incompleteness and inconsistency within NL requirements deliverables. In the SWPDC case study, in some situations the test designer had to insert new "extra" requirements in order to generate coherent Executable Test Cases. These requirements had not been explicited in any of the NL deliverables developed in the scope of the QSEE project, and consulted for generation of the models. We intend to address this problem by means of model checking (Baier and Katoen, 2008).

Acknowledgements We would like to thank and dedicate this research to Prof. José Demísio Simões da Silva (in memoriam) for his relevant contributions to the better development of our research.

References

Abrial JR (2006) Formal methods in industry: achievements, problems, future. In: Proceedings of the 28th International Conference on Software Engineering (ICSE), ACM, New York, NY, USA, pp 761–768

Ambriola V, Gervasi V (1997) Processing natural language requirements. In: Proceedings of the 12th International Conference on Automated Software Engineering (ASE), IEEE Computer Society, Washington, DC, USA, pp 36–45

Ambriola V, Gervasi V (2006) On the systematic analysis of natural language requirements with CIRCE. Automated Software Engineering 13(1):107–167

Ambrósio AM, Mattiello-Francisco F, Santiago VA, Silva WP, Martins E (2007) Designing fault injection experiments using state-based model to test a space software. In: Dependable computing, LNCS, vol 4746, Springer Berlin/Heidelberg, Berlin/Heidelberg, Germany, pp 170–178

Antoniol G, Briand LC, Di Penta M, Labiche Y (2002) A case study using the round-trip strategy for state-based class testing. In: Proceedings of the 13th IEEE International Symposium on Software Reliability Engineering (ISSRE), IEEE Computer Society, Washington, DC, USA, pp 269–279

Baier C, Katoen JP (2008) Principles of model checking. The MIT Press, Cambridge, MA, USA, 975 p.

Balcer M, Hasling W, Ostrand T (1989) Automatic generation of test scripts from formal test specifications. ACM SIGSOFT Software Engeneering Notes 14(8):210–218

Basili VR, Green S, Laitenberger O, Lanubile F, Shull F, Sørumgård S, Zelkowitz MV (1996) The empirical investigation of Perspective-Based Reading. Empirical Software Engineering Journal 1(2):133–164

Bertolino A, Gnesi S (2003) Use case-based testing of product lines. ACM SIGSOFT Software Engineering Notes 28(5):355–358

Binder RV (1999) Testing object-oriented systems. Addison-Wesley Professional, USA, 1248 p.

Bresciani P, Perini A, Giorgini P, Giunchiglia F, Mylopoulos J (2004) Tropos: an agent-oriented software development methodology. Autonomous Agents and Multi-Agent Systems 8(3):203–236

Briand LC, Labiche Y (2002) A UML-based approach to system testing. Journal of Software and Systems Modeling 1(1):10–42

Briand LC, Labiche Y, Wang Y (2004) Using simulation to empirically investigate test coverage criteria based on Statechart. In: Proceedings of the 26th International Conference on Software Engineering (ICSE), IEEE Computer Society, Washington, DC, USA, pp 86–95

Chow TS (1978) Testing software design modeled by finite-state machines. IEEE Transactions on Software Engineering SE-4(3):178–187

Clarke EM, Lerda F (2007) Model checking: software and beyond. Journal of Universal Computer Science 13(5):639–649

Cristiá M, Monetti P (2009) Implementing and applying the Stocks-Carrington framework for model-based testing. In: Breitman K, Cavalcanti A (eds) Formal Methods and Software Engineering, LNCS, vol 5885, Springer Berlin/Heidelberg, Berlin/Heidelberg, Germany, pp 167–185

Cristiá M, Santiago V, Vijaykumar NL (2010) On comparing and complementing two MBT approaches. In: Proceedings of the 11th Latin-American Test Workshop (LATW), IEEE Computer Society, Washington, DC, USA, pp 1–6

ECSS (2008) ECSS-S-ST-00C: ECSS system - Description, implementation and general requirements. European Cooperation for Space Standardization (ECSS), Noordwijk, The Netherlands, 34 p.

ECSS (2009) ECSS-E-ST-10-06C: ECSS Space engineering - Technical requirements specification. European Cooperation for Space Standardization (ECSS), Noordwijk, The Netherlands, 31 p.

El-Far IK, Whittaker JA (2001) Model-based software testing. In: Marciniak JJ (ed) Encyclopedia of software engineering, Wiley, USA

Fantechi A, Spinicci E (2005) A content analysis technique for inconsistency detection in software requirements documents. In: Proceedings of the VIII Workshop on Requirements Engineering (WER), pp 245–256

Fantechi A, Gnesi S, Lami G, Maccari A (2003) Applications of linguistic techniques for use case analysis. Requirements Engineering 8(3):161–170

Fraser MD, Kumar K, Vaishnavi VK (1991) Informal and formal requirements specification languages: bridging the gap. IEEE Transactions on Software Engineering 17(5):454–466

Frohlich P, Link J (2000) Automated test case generation from dynamic models. In: ECOOP 2000: object-oriented programming, LNCS, vol 1850, Springer Berlin/Heidelberg, Berlin/Heidelberg, Germany, pp 472–491

Fuchs NE, Schwertel U, Schwitter R (1999) Attempto Controlled English - not just another logic specification language. In: Logic-based program synthesis and transformation, LNCS, vol 1559, Springer Berlin/Heidelberg, Berlin/Heidelberg, Germany, pp 1–20

Fuchs NE, Schwertel U, Torge S (2000) A natural language front-end to model generation. Journal of Language and Computation 1(2):199–214

Gervasi V, Zowghi D (2005) Reasoning about inconsistencies in natural language requirements. ACM Transactions on Software Engineering and Methodology 14(3):277–330

Harel D (1987) Statecharts: a visual formalism for complex systems. Science of Computer Programming 8:231–274

Harel D, Pnueli A, Schmidt JP, Sherman R (1987) On the formal semantics of Statecharts (extended abstract). In: Proceedings of the 2nd IEEE Symposium on Logic in Computer Science (LICS), IEEE Computer Society, Washington, DC, USA, pp 54–64

Hierons RM (1997) Testing from a Z specification. The Journal of Software Testing, Verification and Reliability 7(1):19–33

Hierons RM, Bogdanov K, Bowen JP, Cleaveland R, Derrick J, Dick J, Gheorghe M, Harman M, Kapoor K, Krause P, Lüttgen G, Simons AJH, Vilkomir S, Woodward MR, Zedan H (2009) Using formal specifications to support testing. ACM Computing Surveys 41(2):1–76

Hoare CAR (1985) Communicating Sequential Processes. Prentice-Hall, Englewood Cliffs, NJ, USA, 238 p.

Hopcroft JE, Ullman JD (1979) Introduction to automata theory, languages, and computation. Addison Wesley, Reading, MA, USA, 418 p.

Howden WE (1976) Reliability of the path analysis testing strategy. IEEE Transactions on Software Engineering SE-2(3):208–215

IEEE (1990) IEEE Std 610.12-1990: IEEE Standard Glossary of Software Engineering Terminology. The Institute of Electrical and Electronics Engineers (IEEE), New York, NY, USA, 83 p.

Jiang JJ, Conrath DW (1997) Semantic similarity based on corpus statistics and lexical taxonomy. In: Proceedings of the 10th International Conference Research on Computational Linguistics (ROCLING), pp 19–33

Kim HY, Sheldon FT (2004) Testing software requirements with Z and Statecharts applied to an embedded control system. Software Quality Journal 12(3):231–264

Leacock C, Chodorow M (1998) Combining local context and WordNet similarity for word sense identification. In: Fellbaum C (ed) WordNet: an electronic lexical database, The MIT Press, Cambridge, MA, USA, chap 11, pp 265–283

Lee D, Yannakakis M (1996) Principles and methods of testing finite state machines: a survey. Proceedings of the IEEE 84(8):1090–1123

Lei Y, Tai KC (1998) In-Parameter-Order: A test generation strategy for pairwise testing. In: Proceedings of the 3rd IEEE International Symposium on High-Assurance Systems Engineering (HASE), IEEE Computer Society, Washington, DC, USA, pp 254–261

Lei Y, Kacker R, Kuhn DR, Okun V, Lawrence J (2007) IPOG: A general strategy for t-way software testing. In: Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS), IEEE Computer Society, Washington, DC, USA,

pp 549–556
Lesk M (1986) Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In: Proceedings of the 5th International Conference on Systems Documentation (SIGDOC), ACM, New York, NY, USA, pp 24–26
Liang J, Palmer JD (1994) A pattern matching and clustering based approach for supporting requirements transformation. In: Proceedings of the 1st IEEE International Conference on Requirements Engineering (ICRE), IEEE Computer Society, Washington, DC, USA, pp 180–183
Lu CW, Chang CH, Chu WC, Cheng YW, Chang HC (2008) A requirement tool to support model-based Requirement Engineering. In: Proceedings of the 32nd Annual IEEE International Computer Software and Applications Conference (COMPSAC), IEEE Computer Society, Washington, DC, USA, pp 712–717
Marcus MP, Marcinkiewicz MA, Santorini B (1993) Building a large annotated corpus of English: The Penn Treebank. Computational Linguistics 19(2):313–330
Masiero PC, Maldonado JC, Boaventura IG (1994) A reachability tree for Statecharts and analysis of some properties. Information and Software Technology 36(10):615–624
Mathur AP (2008) Foundations of software testing. Dorling Kindersley (India), Pearson Education in South Asia, Delhi, India, 689 p.
Mich L (1996) NL-OOPS: from natural language to object oriented requirements using the natural language processing system LOLITA. Natural Language Engineering 2(2):161–187
Mich L, Franch M, Inverardi P (2004) Market research for requirements analysis using linguistic tools. Requirements Engineering Journal 9(1):40–56
Miller GA (1998) Nouns in WordNet. In: Fellbaum C (ed) WordNet: an electronic lexical database, The MIT Press, Cambridge, MA, USA, chap 1, pp 23–46
Miller GA, Leacock C, Tengi R, Bunker RT (1993) A semantic concordance. In: Proceedings of the Workshop on Human Language Technology (HLT), Association for Computational Linguistics, Morristown, NJ, USA, pp 303–308
Morgan R, Garigliano R, Callaghan P, Poria S, Smith M, Urbanowicz A, Collingham R, Costantino M, Cooper C, LOLITA Group (1995) University of Durham: description of the LOLITA system as used in MUC-6. In: Proceedings of the 6th Message Understanding Conference (MUC-6), pp 71–85
Navigli R (2009) Word sense disambiguation: A survey. ACM Computing Surveys 41(2):1–69
Offutt J, Abdurazik A (1999) Generating tests from UML specifications. In: UML'99: the Unified Modeling Language, LNCS, vol 1723, Springer Berlin/Heidelberg, Berlin/Heidelberg, Germany, pp 416–429
OMG (2007) OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2. The Object Management Group (OMG), Needham, MA, USA, 722 p.
Ostrand TJ, Balcer MJ (1988) The category-partition method for specifying and generating functional tests. Communications of the ACM 31(6):676–686
Paradkar A (2003) Towards model-based generation of self-priming and self-checking conformance tests for interactive systems. In: Proceedings of the 18th ACM Symposium on Applied Computing (SAC), ACM, New York, NY, USA, pp 1110–1117
Pedersen T, Patwardhan S, Michelizzi J (2004) WordNet::Similarity: measuring the relatedness of concepts. In: Proceedings of the 5th Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL), Association for Computational Linguistics, Morristown, NJ, USA, pp 38–41
Petrenko A, Yevtushenko N (2005) Testing from partial deterministic FSM specifications. IEEE Transactions on Computers 54(9):1154–1165

Pimont S, Rault JC (1976) A software reliability assessment based on a structural and behavioral analysis of programs. In: Proceedings of the 2nd International Conference on Software Engineering (ICSE), ACM, New York, NY, USA, pp 486–491

Pressman RS (2001) Software Engineering: a practitioner's approach, 5th edn. McGraw-Hill, New York, NY, USA, 860 p.

Santiago V, Amaral ASM, Vijaykumar NL, Mattiello-Francisco MF, Martins E, Lopes OC (2006) A practical approach for automated test case generation using Statecharts. In: Proceedings of the 30th Annual International Computer Software & Applications Conference (COMPSAC) - International Workshop on Testing and Quality Assurance for Component-Based Systems (TQACBS), IEEE Computer Society, Los Alamitos, CA, USA, pp 183–188

Santiago V, Mattiello-Francisco F, Costa R, Silva WP, Ambrosio AM (2007) QSEE project: an experience in outsourcing software development for space applications. In: Proceedings of the 19th International Conference on Software Engineering & Knowledge Engineering (SEKE), Knowledge Systems Institute Graduate School, Skokie, IL, USA, pp 51–56

Santiago V, Silva WP, Vijaykumar NL (2008a) Shortening test case execution time for embedded software. In: Proceedings of the 2nd IEEE International Conference on Secure System Integration and Reliability Improvement (SSIRI), IEEE Computer Society, Washington, DC, USA, pp 81–88

Santiago V, Vijaykumar NL, Guimaraes D, Amaral AS, Ferreira E (2008b) An environment for automated test case generation from Statechart-based and Finite State Machine-based behavioral models. In: Proceedings of the 1st International Conference on Software Testing, Verification and Validation (ICST) - Workshop on Advances in Model Based Testing (A-MOST), IEEE Computer Society, Washington, DC, USA, pp 63–72

Santiago Júnior VA, Cristiá M, Vijaykumar NL (2010) Model-based test case generation using Statecharts and Z: a comparison and a combined approach. INPE, São José dos Campos, URL http://urlib.net/sid.inpe.br/mtc-m19@80/2010/02.26.14.05, (INPE-16677-RPQ/850)

Sarma M, Mall R (2009) Automatic generation of test specifications for coverage of system state transitions. Information and Software Technology 51(2):418–432

Sidhu DP, Leung TK (1989) Formal methods for protocol testing: a detailed study. IEEE Transactions on Software Engineering 15(4):413–426

Sinha R, Mihalcea R (2007) Unsupervised graph-based word sense disambiguation using measures of word semantic similarity. In: Proceedings of the International Conference on Semantic Computing (ICSC), IEEE Computer Society, Washington, DC, USA, pp 363–369

Sleator DD, Temperley D (1993) Parsing English with a link grammar. In: Proceedings of the 3rd International Workshop on Parsing Technologies, pp 277–292

Sneed HM (2007) Testing against natural language requirements. In: Proceedings of the 7th International Conference on Quality Software (QSIC), IEEE Computer Society, Washington, DC, USA, pp 380–387

Souza ÉF (2010) Geração de casos de teste para sistemas da área espacial usando critérios de teste para máquinas de estados finitos. Instituto Nacional de Pesquisas Espaciais, São José dos Campos, SP, Brazil, Master Dissertation, 133 p.

Souza SRS (2000) Validação de especificações de sistemas reativos: definição e análise de critérios de teste. Universidade de São Paulo, São Carlos, SP, Brazil, PhD Thesis, 264 p.

Spivey JM (1989) The Z notation: a reference manual. Prentice-Hall, Upper Saddle River, NJ, USA

Toutanova K, Klein D, Manning CD, Singer Y (2003) Feature-rich part-of-speech tagging with a cyclic dependency network. In: Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology, pp 173–180

University of Ottawa (2008) Alan Williams' page. Available from: http://www.site.uottawa.ca/˜awilliam/, access in: July 22, 2010

University of Sussex (2010) David Hope's page. Available from: http://www.cogs.susx.ac.uk/users/drh21/, access in: July 22, 2010

Vijaykumar NL, Carvalho SV, Francês CRL, Abdurahiman V, Amaral ASM (2006) Performance evaluation from Statecharts representation of complex systems: Markov approach. In: Anais do XXVI Congresso da Sociedade Brasileira de Computação (CSBC) - Workshop em Desempenho de Sistemas Computacionais e de Comunicação, Sociedade Brasileira de Computação, Porto Alegre, RS, Brazil, pp 183–202