J Braz Comput Soc (2013) 19:325–339
DOI 10.1007/s13173-013-0104-z

ORIGINAL PAPER

Functional test data generation for Simulink-like models

Rodrigo Fraxino Araujo · Marcio Eduardo Delamaro · Jose Carlos Maldonado

Received: 6 July 2012 / Accepted: 26 February 2013 / Published online: 20 March 2013
© The Brazilian Computer Society 2013

Abstract Embedded systems are increasingly present in many electronic devices and is often related to critical applications. Therefore, the need for a well planned and executed testing procedure is even higher. We intend to contribute in this area by presenting an experimental evaluation of the pairwise combinatorial approach as a technique for test data generation applied specifically to Simulink-like models. In particular, we have applied our strategy to the generated source code of several models. Furthermore, a testing tool was developed to assist in the test data generation process. We show that there is no statistical significant advantages of the proposed approach over random generation of test data, but when used together they yield better results. The feasibility of the experimental results indicate that efforts can be employed in order to obtain a testing strategy integrated within a testing environment.

Keywords Embedded system · Pairwise testing · Simulink · Scicos

1 Introduction

As a result of technological advances, more and more mechanical systems are being replaced by electromechanical ones. This can be noted in the growing number of embedded systems present in cars, aircrafts, trains and electronic devices. Many of these systems are critical and cannot tolerate failures. Therefore, the testing of embedded systems is a very important task [23].

The National Institute of Standards and Technology estimates that, in the United States, the cost for insufficiency in the software testing process in 2000 was around 59 billion dollars. An example is what frequently happens with automotive models. Recall is a common practice to correct faults that have been introduced during the manufacturing of embedded systems [6].

Due to the complexity of systems and the ever-increasing needs for shortening time-to-market pressures, the testing task has become even more challenging. A common problem is the testing stage being performed at the end of a project development life cycle. Thus, when faults are found, the cost to fix them is much higher [34].

Furthermore, although the implementation of automated testing activity is a common practice, the creation of test sets is still performed manually in many cases. Embedded systems, with increasingly sophisticated software, may have a high number of combinations of inputs and events, which can result in many different outputs, leading to a possible failure to cover all outcomes in a manual testing activity.

A possibility to lessen the aforementioned problem is by using precise models that support a system development life cycle. Models are concise and understandable abstractions that capture the decisions of the functions of a system whose semantics are derived from the concepts and theories of a specific domain [32].

In this scenario, platforms such as ScicosLab/Scicos [24] and Matlab/Simulink [44] are widely used to design and simulate embedded system models. One of their advantages is the application analysis at different levels of abstraction.

R. F. Araujo (✉) · M. E. Delamaro · J. C. Maldonado
Instituto de Computação e Matemática Computacional, Universidade de São Paulo, São Carlos, Brazil
e-mail: rfaraujo@icmc.usp.br

M. E. Delamaro
e-mail: delamaro@icmc.usp.br

J. C. Maldonado
e-mail: jcmaldon@icmc.usp.br

R. F. Araujo
Linux Technology Center, IBM, São Paulo, Brazil

Another benefit is the automatic code generation, which reduces development costs and programming errors.

To ensure the reliability of this kind of system, the industry has been investing in an approach known as model based testing [8]. In this approach, it is easier to automate the testing activity, which includes an automatic generation of test sets. The testing activity takes place at a more abstract level, even before the software is coded. This leads to a more efficient process with significant cost reduction and a final product with higher quality.

Seeking to address such issue, our main results come from the experimental studies that were conducted to assess the adequacy of test data generation using two different approaches, a pairwise generation [19] and a random one. The adequacy assessment of the test data was performed by applying the mutation testing in a small set of programs and comparing the mutation score achieved by each test data generation approach.

For the conduct of such experiments, we introduce a tool called TeTooDS (Testing Tool for Embedded Systems) [3], which assists in the test data generation by applying the pairwise approach in embedded systems models. This approach ensures that any two possible values, belonging to two different parameters, will be present in at least one test data [19].

Kuhn et al. [26] show that the combinatorial design approach for automatic test data generation is quite effective in many situations and that the pairwise testing is fit for most general applications. Thereby, we considered that efforts could be employed by applying this approach in the context of embedded systems models. We are also not aware of other studies that focus on a well planned and documented experimental evaluation of test data generation methods for Simulink-like models.

This paper is structured as follows. Section 2 presents some characteristics of the environments for embedded systems development and simulation. In Sect. 3, testing techniques for Simulink-like models are described. The pairwise approach and the testing tool we developed to assist in the testing of Simulink-like models are detailed in Sect. 4. Section 5 presents the experimental results of the test data generation methods applied to a small set of models. The concluding remarks, possible extensions for the testing tool and suggestions for further work are presented in Sect. 6.

2 Development and simulation environments

Embedded systems devices are an increasingly higher portion of several technological areas. In the embedded systems development, it is common to use simulation prior to hardware and software integration. This occurs mainly because the hardware may not be available for testing, or may even not exist. The simulation can also avoid dangerous situations, such as equipment damage or human lack of safety [21,23].

In this context [2], emphasize that a dynamic system consists of a set of possible states, together with a rule that determines the present state from a past state. According to [25], dynamic systems relate model-system states to earlier states. Classical physics, for example, predicts continuous changes of quantities such as position, velocity, or voltage with continuous time.

Simulink [44], a software that works with Matlab, and Scicos [24], a software that works with ScicosLab, are alternatives for the development and simulation of dynamic systems. One of their advantages is the application analysis possibility in several levels of abstraction. Another benefit is the automatic generation of code, thereby reducing development costs and programming errors.

Such systems are mathematically represented by systems of equations, that are differential equations in the case of continuous time systems, difference equations in the case of discrete time systems, and a mix of both in the case of hybrid systems. The simulation of these type of systems is based on numerical algorithms, whereby the solution of a system of equations, i.e., the semantics of a Simulink model, is given by the sequence of values that represents the temporal functions [11].

2.1 Simulink

Simulink is software for modeling and simulating embedded systems or, more precisely, dynamic systems. It provides a common environment for sharing data, designs and specifications, making it possible to develop more reliable critical systems and generating code with security [43].

Large worldwide organizations make use of Simulink. One of the main areas is the aerospace industry. The Airbus A380, the Mars Exploration Rover and the F-35 joint strike fighter were modeled using Simulink [44]. Embraer, a Brazilian exponent company in aviation has extensively used Simulink as shown, for instance, in Cavalcanti and Papini [10].

These models are based on blocks diagrams. These blocks include a library of sinks, sources, connectors and linear and non-linear components. Moreover, it is possible to create blocks for specific purposes. Models can be hierarchical and it is possible to analyze the whole system on a high level or detail each one of the blocks, increasing the model abstraction level. It facilitates the understanding of the model organization and on how the components interact with each other [36,43].

2.2 Scicos

Scicos is also software for modeling and simulating embedded systems [24]. Unlike Simulink, Scicos is free software. For modeling, it offers a modular way to build complete embedded systems by an editor of block diagrams. It is possible to build a library of reusable modules (blocks) that can be used in different systems and projects [9]. A large number of blocks, that can be used to perform basic operations, are already present on the platform. It is almost never necessary to build a block from scratch.

Moreover, as with Simulink, Scicos provides features that help in the optimization, validation and generation of C code for a particular model. For example, an application can have its development cost decreased by traditional optimization techniques, validated by simulation and its C code generated for specific hardware.

2.3 Example

Simulink-like models are composed by blocks connected by lines (signals). These blocks can be elementary, containing simple operations (as arithmetics, for instance), or subsystems, that contains a composition of elementary blocks. It is worth emphasizing the *Integrator* and the *UnitDelay* blocks, which introduce the notion of time. When an Integrator is used, the model is said to be of continuous time, and the operation associated with the block is a mathematical integration over time. A model that uses a UnitDelay is said to be of discrete time. A mix of both produces a hybrid model, defined as a data flow where the signals are continuous or discrete time functions.

Figure 1 contains an example of a Simulink-like model that is divided into three subsystems [11]. A continuous time subsystem is present in Fig. 1a and represents a braking pedal as a mass-spring-damper mechanical system. A discrete time subsystem is present in Fig. 1b and is responsible for detecting when the pressing force is greater than a given threshold to activate the brake. Fig. 1c presents the main system, a composition of both subsystems, containing an input, the force, and an output, the detection result.

3 Embedded system testing

Embedded system testing is considered a vital task, especially because many systems are critical. However, some factors may hamper the testing of such systems. If a specific hardware is required, for instance, this test can be prevented by the equipment cost. Another concern is related to the possibility of the hardware being damaged, since the system may not present the expected behavior.

Developers typically use simulation prior to the integration of hardware and software. However, the simulation may also present some problems, e.g., a generation of potentially very large data sets, a high consumption of time, error prone outputs and temporal restrictions [22].

We conducted a systematic review that aimed to identify methods for the automatic generation of test data from embedded system models. Among the 29 selected papers, 15 handle specifically Simulink-like models [4]. We could notice that most of the papers present or make use of commercial testing tools and that there are no studies regarding the test of Scicos models, probably because Simulink is more consolidated within industry.

Concerning specifically the testing of Simulink-like models [22], focus on the use of functional characteristics to test this type of model. A great deal of the studies [1,5,12,17,18,37,39,41,44,46,48] involve the application of structural techniques, which can be motivated by guidelines such as DO-178B [38], that requires minimum structural coverage assurances. The mutation testing can also be used, as described by [7,47,48]. A more detailed description of each study is presented as follows.

The functional testing is a technique used to design test cases in which a program or system is considered as a black box, and to test it, inputs are provided and the outputs generated are evaluated to verify their compliance with the specified goals. In principle, the functional testing can detect all faults, submitting the program or system to all possible inputs. However, the input domain may be infinite or very large, making the test time of an activity unfeasible, and thus rendering this alternative not practical [33].

Due to the possible large number of inputs and outputs, a tool that supports the automatic generation of these values can help the tester's task. The test values must be created for each input variable on each time step throughout the simulation. In addition, to help with the test data generation, a testing tool can capture, organize and display the generated output for the tester analysis [22].

Henry et al. [21,22] propose the use of MATT (Matlab Automated Testing Tool), a tool that focuses on real time systems testing. It helps with the generation of test data that exercise a specified system in different conditions. According to the authors, it is necessary to use critical values, system dependent, values between the minimum and maximum bounds, and input values that are outside of the specified domain. The approach they present helps to generate test data which take the system to such conditions.

Blackburn and Busser [5] developed T-VEC [42], a commercial tool that integrates the development and testing based on functional requirements. It uses a requirements definition model, and tracks all of the requirements to a embedded system model or code, ensuring that each specified requirement is tested. It is used by large organizations such as Lockheed Martin [42]. One of the important features of the tool is the automatic generation of test vectors. It has the ability to determine the inputs, the expected output, and a mapping of each test case to their functional requirement.

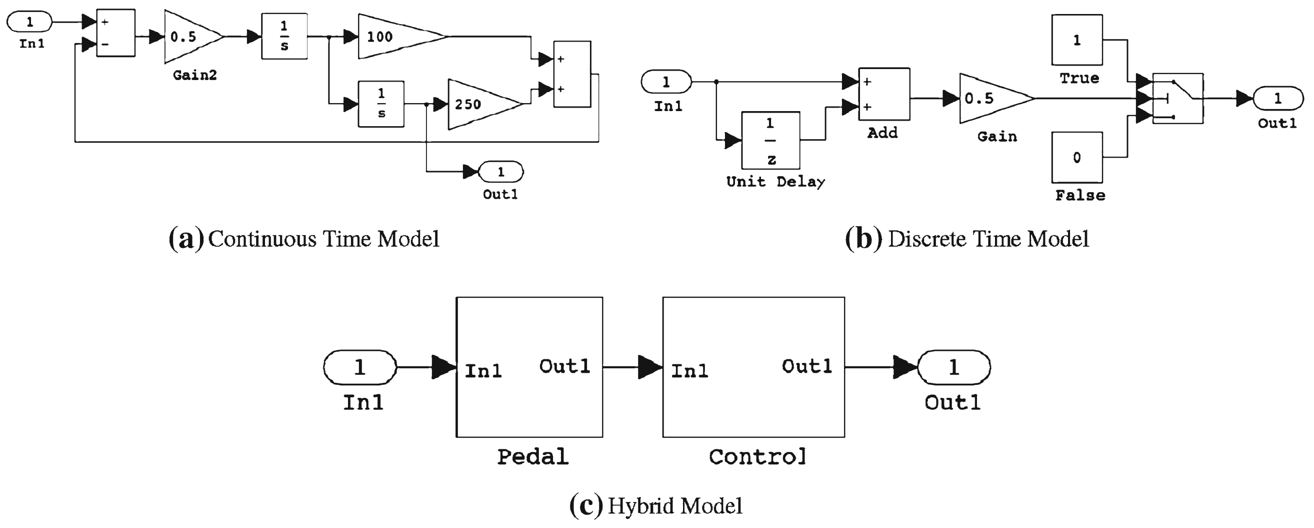

Fig. 1 Simulink-like models

Another possibility is to use the structural technique for testing a given system, which requires the implementation of pieces or components of a program. The logical paths are tested by the selection of test data that exercise a specific condition set and pairs of definition and use of the variables [33].

In general, most of the structural criteria make use of a representation known as Control Flow Graph (CFG). A program can be decomposed into a set of disjointed blocks of commands. The execution of the first command of a block results in the execution of all other block commands. All commands in a block, except possibly the first, have one predecessor and exactly one successor, except possibly the last command [31,35].

Zhan and Clark [46,48] describe an alternative for the test data generation using the structural technique. The solution consists of first performing a random test data generation, aiming to improve the operation cost. Subsequently, an analysis of the paths which have already been traversed is used for the generation of the remaining test data. This proposal is based on the costs to cover each possible path. Probes are inserted in the second input of each switch block, a block that can be compared to an if-then-else sentence. Thus, it is possible to analyze which values are being supplied and for each condition identified, it is possible to associate a cost indicating how far the data is to go to satisfy the switch block condition.

Gadkari et al. [17] present the AutoMOTGen, a tool for the automatic generation of test data from Simulink models in order to test automotive controllers. The goal is to use the Simulink model, high-level requirements and test specifications to apply model checking techniques for the generation of test data. Model checking techniques aim to automatically test whether a model meets a given specification. The tool uses the SAL language to create an intermediate model, by converting the Simulink model into a formal model of finite states, translating the specifications and the high level testing requirements into formal properties by the use of linear temporal logic (LTL). This model is structured to contain hierarchical information and a mapping of the structural coverage of the Simulink model in relation to the testing requirements.

Satpathy et al. [39] propose a technique and testing tool for Simulink models called Randomized DIRECTED Testing (REDIRECT). It is based on four principles: test by using random inputs, direct traversion of a block previously reached, backtracking and random test based on feedback. One of the highlights of the tool is that REDIRECT uses a heuristic guided by defined patterns and can reach non-linear blocks, whose output is not proportional to their input.

Mathworks released Simulink Design Verifier [44], a tool that generates test values by the use of formal analysis techniques to achieve exhaustive evaluation of a Simulink model. The technique is based on mathematically rigorous procedures to simplify and search through the possible execution paths of a model. Its advantages include the detection of incomplete requirements and the exploration of design faults. Nevertheless, not all Simulink features are supported by the tool.

Reactis [37,41] is a commercial testing tool that can generate test sets to exercise a Simulink model. For the generation of test data, a random generation is performed and the inputs are selected by Monte Carlo methods. The guided simulation technique is also used, in which values are analyzed and chosen in order to cover the remaining test requirements of the model. This test set covers the MC/DC (Modified Conditions/Coverage Decisions) criterion. The tool contains three main components: Tester, which automatically generates test sets; Simulator, which allows the visualization of the execution; and validator, that performs a search for violations in the requirements specified by the user [12].

Beacon Tester [1] is also a commercial tool that automatically creates test vectors in order to ensure the quality and reusability of a Simulink model. The coverage obtained with the test vectors include functional criteria, such as boundary value analysis, and structural criteria, such as all-nodes, all-edges and MC/DC.

One more commercial tool dedicated to the generation of test data for Simulink models is Safety Test Builder [18]. At first, the generation of test data is random, and then a heuristic algorithm is used. The test set generated aims to cover structural criteria such as MC/DC. Very little documentation can be found regarding Beacon Tester and Safety Test Builder.

The mutation testing can be an alternative for the testing of Simulink-like models. In the mutation criterion typical implementation faults are used to generate testing requirements. The program being tested is altered several times, creating a set of alternative programs (mutants). The tester is responsible for choosing test data that show difference in the behavior among the original program and the mutant programs [31]. The test set quality is measured according to its ability to detect faults in the mutants [15].

In Zhan and Clark [47,48], a solution for the generation of test data using the mutation testing is described. It is necessary to perform a random generation of test data for a model. For the mutants that have not been differentiated from the original program, the proposal is based on faults propagation costs inserted by mutation operators to the model outputs. If a test data is too weak to propagate a fault, that is, if the measurement that is done shows that this test data is far from acting on the mutant part of the code and influence the output, it receives a high cost. If a test data is good for spreading a particular fault to the output, it receives a low cost. One of the possible weaknesses of this proposal is the low numbers of mutation operators defined, i.e., add, multiply and assign.

Brillout et al. [7] developed a methodology to assess the correctness of Simulink models by automating the test data generation activity. Their objective is to cover the requirements imposed by the mutation testing. In order to generate and optimize the test data, the approach focus is on model checking techniques. However, the authors do not present a solution for how to apply the mutation testing, i.e., which mutant operators should be used to generate the testing requirements.

Generally, different techniques are complementary and should be combined in practice. In the next section, our approach for applying the pairwise testing in embedded system models is described.

4 Combinatorial testing for Simulink-like models

In this section, TeTooDS, a tool that supports the testing of Simulink-like models, is presented together with the pairwise testing. The testing tool assists in the extraction of relevant information for the appropriate generation of test data using the pairwise approach. Thereby, we can achieve a reduction in the number of test generated data, improving the computational cost and the effort required to analyze the output data.

The pairwise technique is detailed in Sect. 4.1 and TeTooDS is described in Sect. 4.2.

4.1 Pairwise testing

Functional testing considers the system as a black box from which only inputs and outputs are known. Testing criteria assist in determining inputs for testing the underlying system as well as combining them in such a way that they effectively exercise it.

Critical systems must have a very small probability of failure. Nevertheless, it is difficult to reduce the risk of unpredictable behavior of an embedded system to zero [30]. Therefore, the test data generation activity is of great importance, since the effectiveness of a test criterion depends on the selection of significant values that satisfy existing test requirements. However, the cost for the test data generation can be high and, thus, guidelines must be followed to reduce the required effort.

One possible approach is to apply the pairwise testing, that can generate efficient test sets that contribute to a reduction in the cost of finding adequate test data. An input set is introduced and then an evaluation is performed to check whether the result conforms with the requirements [19].

The combinatorial testing was originally proposed in order to reduce the number of test data required to verify the interoperability among the functions of a system, based on a combinatorial method used in mathematical constructions for statistical experiments [13,20]. Moreover, it presents good code coverage and ability to detect failures [26]. The number of tests needed to cover n combinations of input parameters grows logarithmically according to the number of parameters. It is important to note that the key to minimize the number of test data is the fact that each one covers different combinations (pairs, triples or n-tuples).

The parameters of an example system extracted from Lott et al. [29] are presented in Table 1a. By them it is possible to create 24 values combinations. If the pairwise combinatorial approach is used, only 12 cases would be needed to cover all parameters pairs, as shown in Table 1b. The test set created has the characteristic that given any pair of fields (columns), all possible combinations of values for them are present.

According to Schroeder et al. [40], several studies have been conducted on coverage achievement by pairwise testing. Coverage in the majority of the studies refers to code coverage, or the measurement of the number of lines, branches, decisions or paths of code executed by a particular test suite.

Table 1 Pairwise test

(a) Simple system relations

	Fields		
	Operat. sys.	Stor. sys.	Disp. sys.
Values	GNU/Linux	IDE	Simple
	Win 2000	RAID	AGP 64mb
	Win XP	SCSI	
		Firewire	

(b) Generated test data

Number	Operat. sys.	Stor. sys.	Disp. sys.
1	Win 2000	Firewire	AGP 64mb
2	GNU/Linux	Firewire	Simple
3	Win 2000	SCSI	Simple
4	GNU/Linux	IDE	AGP 64mb
5	Win 2000	RAID	Simple
6	Win XP	IDE	Simple
7	GNU/Linux	RAID	AGP 64mb
8	Win XP	SCSI	AGP 64mb
9	GNU/Linux	SCSI	AGP 64mb
10	Win 2000	IDE	Simple
11	Win XP	RAID	AGP 64mb
12	Win XP	Firewire	Simple

High code coverage has been correlated with high fault detection. In general, the conclusion of these studies is that testing efficiency, i.e., the amount of time and resources required to conduct testing, are optimized because the pairwise test set achieves the same level of coverage as larger combinatorial testing sets.

However, the results of these case studies are difficult to generalize. These practices are not described in enough detail to understand the full significance of these results. Additionally, little is known about the characteristics of software systems used in the studies [40].

Taking into account that the pairwise strategy mainly targets functional specifications, we have considered that an investigation of its suitability to Simulink-like models would be appropriate. These kind of models are a design implementation of a functional specification, that is going to be ultimately converted to low level code.

4.2 Tool for pairwise testing

TeTooDS is a testing tool specifically designed to interpret Simulink-like models, interact with simulation environments, and is used to assist in the test data generation task. It has been developed to provide support for the application of functional criteria, specifically the pairwise approach, in Simulink-like models. For that, it is possible to:

– parse the models for relevant features to the application of the functional technique, like the input variables and their types. Based on these features, the tool provides to the tester a set of standard values that can be used as input values for the test, as the inferior and superior limits;
– the tester, aware of the system specification, can change these values by setting, for instance, typical ranges for certain variables or values. It is also possible to use the proposal from Henry et al. [22], allowing the tester to select values that lead to transitions that supposedly exercise the model, or at least all input variables, completely;
– generate the test data using the pairwise approach, producing input test data that can be used in the model simulation in a computer environment or in the C generated code; and
– exhibit the outputs acquired from the external simulation environment.

TeTooDS works by creating testing projects. When one is created, the tool parses relevant information from a Simulink-like model. Such information must be stored in an XML file, that can be generated from a Simulink or Scicos parser, or even manually. The information retrieved include input ports, input datatypes, blocks, connections and output ports. Then, the tester is responsible for selecting how the test data are going to be generated.

To help on the definition of appropriate values, the testing tool provides a data type editor. It is possible to create new data types, based on the limits of already existing ones. It can be useful to select more than one range of values for a data type or to select different ranges of values for inputs that originally have the same data type. It is also possible to select how many values will be selected in an interval and how this generation will be performed (ascending order, descending order or randomly).

For instance, an input X is defined in the model as a *double* data type and represents a temperature in the range of -300 to 300. An input Y is also defined as a *double* data type, but represents a velocity in the range of 0–120. The testing tool permits the creation of a *temperatureDouble* data type with a range of -300 to 300 and a *velocityDouble* data type in the range of 0–120. It is possible to assign these new data types to X and Y. When the test data are generated, these new ranges are used.

The tester also needs to specify how many values are going to be selected for each range and how this selection is going to be performed. For example, the tester can specify that 6 values are going to be selected for the *temperatureDouble* range, in an ascending way. That means that the values -300, -180, -60, 60, 180 and 300 are going to be used in the test data generation. If the random option is chosen, then it is not possible to predict which values of that range are going to be selected.

The test data can be generated from the pairwise approach or randomly. If the first option is chosen, it means that any two different input parameters values are present in at least one test data. Otherwise, there are no guarantees of what combinations exist and it is necessary to specify how many test data should be generated. Additional test data can be added manually, as well as existing test data can be removed.

Several strategies can be used to implement the pairwise generation technique [19]. In particular, the IPO strategy (In Parameter Order) is used by TeTooDS for the test data generation [28]. This strategy is used because its implementation is simple, the algorithm can be extended to support more parameter combinations and its number of generated combinations is similar to other strategies [19,27].

The IPO algorithm first tries to cover the first two model parameters pairs. Then it extends the test data in order to cover the remaining parameters, one at a time. Two basic algorithms are used, one for the horizontal growth, which extends the existing test data to cover a new parameter, and other for the vertical growth, which creates new instances of test data to ensure that the coverage of this new parameter pairs is complete.

After the generation of test data, they can be applied in a simulation or in an executable application via the testing tool. Corresponding relationships between the inputs (test data) and generated outputs will be presented to the tester for further analysis. It is worth mentioning that the testing tool merely interacts with the execution environment to gather the output data.

5 Experimental studies

There are two paradigms that are usually used as the basis for the development of an empirical study. The qualitative paradigm aims at the study of objects in their natural environment. The researcher seeks phenomena interpretation based on explanations. The quantitative one focuses on quantifying a relation or the comparison of two or more groups. The objective is to identify cause and effect relations and is normally related to data collection by case studies [45]. The two paradigms should be considered complementary and not competitive.

For the scope of this paper, the qualitative assessment was aimed through the analysis of the test data generated by TeTooDS in relation to mutation criterion. The quantitative evaluation was also an object of study as we used several models in our experiment.

In this experiment we aimed to measure the test data generated using the pairwise approach against a random generation set of test data. This comparison could be achieved by applying the mutation testing as a mean to obtain the respective scores when applying each test data set to a model.

To help with the task of measuring the coverages obtained, testing tools are of major importance. However, as there are no tools available to work directly on embedded system models, our alternative was to use the C code generated from the models.

Thereby, the test sets generated by TeTooDS were evaluated for each one of the programs by using Proteum [14] to support the mutation testing. In Proteum, all of its 73 mutation operators were used. However, the code generated by Scicos has too many unused variables and other pieces of unexecuted code, resulting in a large number of equivalent mutants to be analyzed (up to 500,000 mutants). It was necessary to use a reduction strategy to the generation of mutants for these models. In that case, 10 % of the mutants were selected per mutation operator.

5.1 Planning

Five Simulink-like models were gathered for this experiment. The Tiny, Quadratic and Duplex models were extracted from Zhan and Clark [48]. The Flow Control model was selected from Blackburn and Busser [5] and the Generic Voter model was provided by Embraer [16].

Although some of the models may seem to perform a small number of operations, we argue that the main nature of Simulink-like models is the implementation of data flow systems in different levels of abstraction.

For each of the models we generated and compared, random test sets combined randomly against random test sets that had their values combined by the pairwise strategy. The purpose of our comparison is to evaluate the effectiveness of each generation strategy measured by the capacity of distinguishing mutants of the programs generated from the models. Figure 2 describes this process.

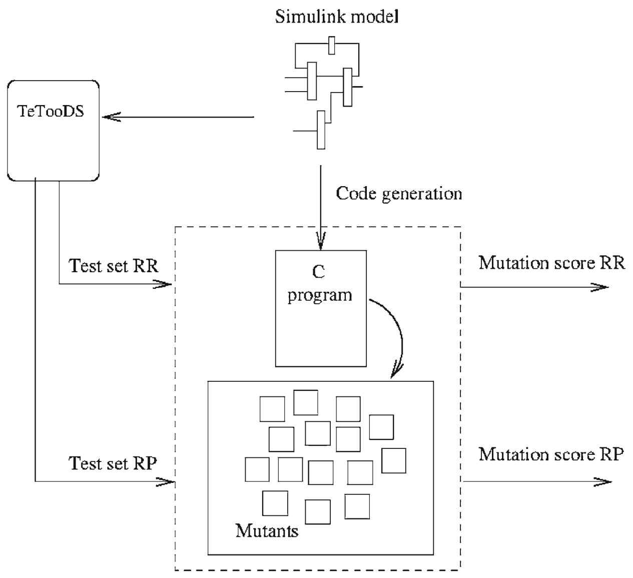

Fig. 2 Test set evaluation

Ranges of values within and out of the input domain of each model were used, and different quantities of values for each interval were specified. Our approach is dependent on the tester for the choice of appropriate intervals of values.

We have generated, according to the design of Fig. 2, 30 test sets for each generation strategy. This is considered the minimum number of repetitions required for the achievement of statistical significance. Each test set is composed of several groups of test data, where a group comprehends a test data for each input of a model. When a test set is applied to a model, its groups of test data are provided consecutively in continuous time steps to the execution environment. Afterwards, the means of the mutation scores are obtained and compared. We have used the Student *t* test to evaluate the significance of the results, that are described for each model in the following sections.

Furthermore, we have generated test sets of two different sizes, which are characterized as the following:

– RP_1: Random generation with pairwise combination. The number of generated elements depends on the model inputs;
– RR_1: Purely random generation. The number of generated elements is the same as RP_1 set;
– RP_2: Random generation with pairwise combination. The number of generated elements depends on the model inputs, but more elements were selected in each interval when compared to RP_1;
– RR_2: Purely random generation. The number of generated elements is the same as RP_2 set.

For each input we have experimented selecting numbers in an incremental way, starting with the lower and upper bound of the valid domain and with two values outside of it. The number of elements selected in RP_2 was the one that meant that no further improvement could be achieved in the mutation score just by selecting more elements from the input datatype interval.

To avoid possible interference of the code generation process with the results, we used three different programs versions generated from the models, on which the mutation testing was applied to assess the generated test sets. This code was generated in three different ways: (1) manually; (2) from a Simulink model, using Mathworks Real Time Workshop (RTW) and; (3) from a Scicos model, using its own code generator. Section 5.3 details the generation procedure for each model.

By conducting this experiment, we plan to compare the means of the mutations scores obtained by the RR_1 and RP_1 sets and by the RR_2 and RP_2 sets. Considering μ_r as the mean obtained by the random sets and μ_p the mean obtained by the random sets with pairwise combination, we have the following hypothesis to verify:

– Null hypothesis H_0: $\mu_r = \mu_p$;
– Alternate hypothesis H_1: $\mu_r \neq \mu_p$;

After taking into consideration the obtained results, which were not statistically significant, we have decided that we should further investigate the comparison of adequacy among the means of the pairwise and the random combination individually against a union between the pairwise and the random combination. For that matter two additional test sets are used in the experiments:

– $RP_1 + RR_1$: Union between RP_1 and RR_1 test sets.
– $RP_2 + RR_2$: Union between RP_2 and RR_2 test sets.

Despite that the pairwise approach had presented better results in most of our experiments, we could not achieve an statistical significance to support this sentence. Therefore, we have investigated the union of both approaches, that have yielded better results. Details of the path we have followed are discussed in Sect. 5.4.6.

We stress that our experiment is actually based on Simulink models because the technique being evaluated is applied on such models. On the other hand, our evaluation of the quality of the test sets is performed at code level, since *(i)* there is no good set of mutant operators for Simulink-like models; *(ii)* no tool to seed faults or perform mutation testing at model level is available; and *(iii)* it is a reasonable assumption to consider the mutants at code level as representative of possible faults at model level, since the models are ultimately converted into code.

It is important to mention that possible time constraints are not relevant to our experiment. Since we are only using the input information from the models, we do not address the features of a Simulink-like model when generating test data using our approach.

5.2 Threats to validity

An important concern regarding the obtained results is its validity level. It is possible to classify the validity threats of an experiment in basically four types: conclusion, internal, construct or external [45].

The internal validity is of high priority, mainly because an important object of study is the relationship between the causes and the outcomes. A possible threat may be the generation procedure of the test data: a single tester (one of the authors) aware of the models behavior defined the parameters for the generation of the test set. If it can influence on the results, on the other hand, it represents the procedure of the functional testing.

We tried to eliminate the threat to the construct validity by including a comparison of the pairwise generation with randomly generated test data, in addition to the mutation scores obtained.

In our experiment, we make use of a set of mutant operators for low level code instead of an specific set for a Simulink-like model, as explained in Sect. 5.1. Despite the reasonable assumption that we could follow this approach since this type of system is converted into low level code, we considered it appropriate to mention it as a threat to validity.

A statistical significance was achieved by generating each test set 30 times, including: *(i)* a pairwise generation with random values, *(ii)* a random generation with random values and *(iii)* both generations together (values are selected by the pairwise approach and by the random approach). Furthermore, different number of values were selected in each of the specified intervals randomly and with a constant variation among them.

Regarding the external validity, the results are difficult to generalize, as many depend on the tester and the used models. A selection of inappropriate values or models with specific features (such as the need for a specific value as one of the inputs) may provide less satisfactory results.

5.3 Experimental setup

This subsection presents the general descriptions of each model that have been used during our experimental evaluation and their domain inputs test generation parameters.

5.3.1 Tiny

This system was introduced by Zhan and Clark [46] and models a problem constructed by the authors. It has three inputs (X, Y and Z) of double datatype and each input has a domain from -100 to 100. When the expression $((Y - Z) * (Z - X) >= 1{,}000) \&\& (Z * Z >= 8{,}950)$ is true, the output is $(X + Y)$. Otherwise, the output is $(Z * Z)$. This model is presented in Fig. 3.

The test sets generated for this model have the following parameters:

- RP_1: For each of the three inputs of the Tiny model two random values were selected in the range of $[-100, 100]$ (valid inputs), and two were selected in the range of $[-1{,}000, -100[\cup]100, 1{,}000]$ (invalid inputs). Thus, selecting a total of four values per input, 16 test data were generated by the pairwise approach;
- RP_2: The difference from the RP_1 is that, for each input range, five values were selected instead of two. Thus, selecting a total of 10 values per input, 108 test data were generated by the pairwise approach;
- RR_1: Values were generated randomly and no strategy for combining the values was adopted. The number of test data created is the same as set RP_1;
- RR_2: Random generation as the previous one but the number of test data was taken from the cardinality of RP_2;

5.3.2 Flow control

This model [42] represents an electronic regulator that has three subsystems: a flow regulator, a temperature sensor and a logic controller. The system has three input ports: temperature, temperature low bound and temperature high bound. When the temperature is below the low bound, a valve is closed, i.e., receives a zero value. When the temperature is above the high bound, a valve is opened, receiving a value of 100. When the temperature is between these limits, the valve position is calculated by the expression $(5.0/3.0) * (temperature - low\_bound)$. The temperature input domain is in the range of $[-100, 300]$. The other inputs should be the constants 120 and 180, respectively.

The test sets generated for this model have the following parameters:

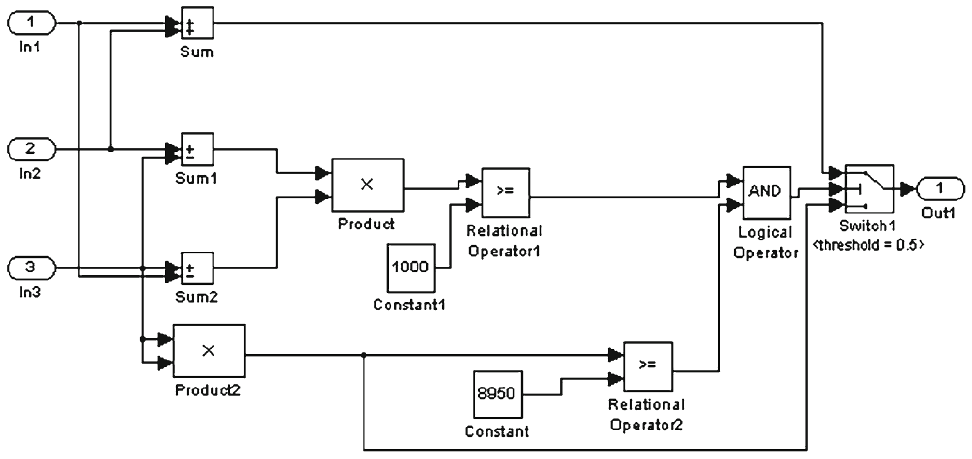

Fig. 3 Tiny model

- RP_1: For the temperature input of the Flow Control model two random values were selected in the range of [−100, 300] (valid inputs), and two were selected in the range of [−1,000, −100[∪]300, 1,000] (invalid inputs). For the temperature low and high bound inputs, we used the values 120 and 180 (valid inputs) and two values outside the domain, i.e., [−1,000, 120[∪]120, 180[∪]180, 1,000] (invalid inputs). Therefore, 13 test data were generated by the pairwise approach;
- RP_2: The difference from the RP_1 is that, for each input range, three values were selected instead of two. Therefore, 23 test data were generated by the pairwise approach;
- RR_1: Values were generated randomly and no strategy for combining the values was adopted. The number of test data created is the same as set RP_1;
- RR_2: Random generation as the previous one but the number of test data was taken from the cardinality of RP_2;

5.3.3 Quadratic

This model [46] has three inputs (X, Y and Z) that have an input domain from −100 to 100 and is presented in Fig. 4. One of the model characteristics is the presence of switch blocks. This block redirects the first or the third input, based on the second input value and can be compared to an if-then-else sentence. The developer must specify how the evaluation of the second input is going to happen. In this model, it is verified if it is greater or equal than a specified threshold.

The test sets generated for this model have the following parameters:

- RP_1: For each of the three inputs of the Quadratic model two random values were selected in the range of [−100, 100] (valid inputs), and two were selected in the range of [−1,000, −100[∪]100, 1,000] (invalid inputs). Thus, selecting a total of four values per input, 16 test data were generated by the pairwise approach;
- RP_2: The difference from the RP_1 is that, for each input range, five values were selected instead of two. Thus, selecting a total of 10 values per input, 38 test data were generated by the pairwise approach;
- RR_1: Values were generated randomly and no strategy for combining the values was adopted. The number of test data created is the same as set RP_1;
- RR_2: Random generation as the previous one but the number of test data was taken from the cardinality of RP_2.

5.3.4 Duplex

This model is a subsystem of a mechanical controller provided by an industrial company [46]. The purpose is to check for faults in an injector flow within a duplex burner. It has eight inputs with a range that goes from −1,000 to 1,000.

The test sets generated for this model have the following parameters:

- RP_1: For each of the eight inputs of the Duplex model two random values were selected in the range of [−1,000, 1,000] (valid inputs) and no values were selected outside of the domain input because no further improvement could be achieved. Thus, 11 test data were generated by the pairwise approach;
- RP_2: The difference from the RP_1 is that, for each input range, three values were selected instead of two. Thus, 23 test data were generated by the pairwise approach;
- RR_1: Values were generated randomly and no strategy for combining the values was adopted. The number of test data created is the same as set RP_1;

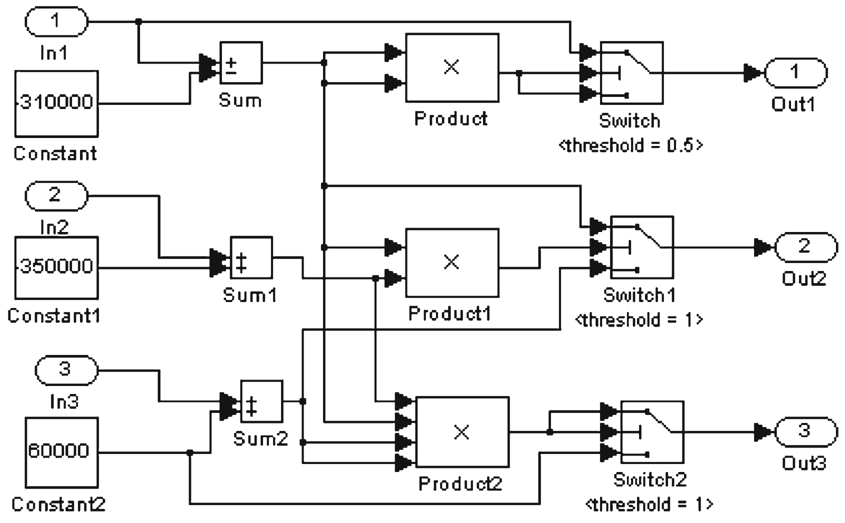

Fig. 4 Quadratic model

Table 2 Tiny

Code	Total mutants	Equivalent mutants	Test data	Generation	Mutation score (%)	p value$_1$	p value$_2$
Manual	1,146	121	16	RP_1	90.7	0.001	0.040
				RR_1	80.5		0
			108	RP_2	95.3	0.263	0.009
				RR_2	95.1		0.021
			32	$RP_1 + RR_1$	93.1		
			216	$RP_2 + RR_2$	95.5		
Simulink	1,666	656	16	RP_1	91.7	0.034	0.043
				RR_1	84.1		0
			108	RP_2	96.3	0.701	0.047
				RR_2	96.3		0.006
			32	$RP_1 + RR_1$	95.0		
			216	$RP_2 + RR_2$	96.6		
Scicos	7,248 (10 %)	5,461	16	RP_1	95.8	0.025	0.005
				RR_1	95.8		0
			108	RP_2	95.9	0.813	0.003
				RR_2	95.8		0.003
			32	$RP_1 + RR_1$	95.9		
			216	$RP_2 + RR_2$	95.9		

– RR_2: Random generation as the previous one but the number of test data was taken from the cardinality of RP_2.

5.3.5 Generic voter

This system was supplied by Embraer, a Brazilian aeronautics company. The main system has seven inputs and three subsystems: Miscompare Detection, Fault Isolation and Average Computation, with a total of 57 blocks. Three of the inputs operate as sensors that measure the same signal, three are of *boolean* data type and may cause the signal to become invalid, and the last one is a threshold input. The signal may become invalid if the difference between any two inputs is greater than the threshold, or if one of the *boolean* inputs is false. The domain of the first three inputs is from −10 to 30 and the threshold input domain is 2.

The test sets generated for this model have the following parameters:

– RP_1: For each of the first three inputs of the Generic Voter model two random values were selected in the range of $[-10, 30]$ (valid inputs), and two were selected in the range of $[-1{,}000, -10[\cup]30, 1{,}000]$ (invalid inputs). Also, boolean values for the next three, and the value 2 (valid input) for the threshold along with two values in the range from $[-1{,}000, 2[\cup]2, 1{,}000]$ (invalid inputs). Thus, 16 test data were generated by the pairwise approach;
– RP_2: The difference from the RP_1 is that, for each input range, five values were selected instead of two. Thus, 108 test data were generated by the pairwise approach;
– RR_1: Values were generated randomly and no strategy for combining the values was adopted. The number of test data created is the same as set RP_1;
– RR_2: Random generation as the previous one but the number of test data was taken from the cardinality of RP_2.

5.4 Experimental results

In this section we present the experimental results obtained with the models described in Section expsetup. The code generated by Simulink and Scicos, like the ones of most of the automation tools, is not optimal. It increased the number of mutants that were generated in comparison with the manual generation of code, that only contains the essence of an embedded system model.

As follows we show detailed information about the results of each model individually. In Sect. 5.4.6, a general discussion of the obtained results is presented.

5.4.1 Tiny

The results from the application of the mutation testing are presented in Table 2. It is possible to realize that the column "Mutation Score" contains the mean of the mutation scores obtained by each test set RP_1, RR_1, RP_2 and RR_2, respectively. The next two lines contains the mutation score of RP_1 together with RR_1 and of RP_2 together with RR_2. The column "p value$_1$" indicates the significance level for the rejection of H_0 and the column "p value$_2$" is the significance obtained by the Student t test of $RP_1 + RR_1$ against RP_1 and RR_1 individually, and of $RP_2 + RR_2$ against RP_2 and RR_2.

Table 3 Flow control

Code	Total mutants	Equivalent mutants	Test data	Generation	Mutation score (%)	p value$_1$	p value$_2$
Manual	1,478	124	13	RP_1	90.7	0.001	0
				RR_1	84.3		0
			23	RP_2	92.6	0.004	0
				RR_2	90.5		0
			26	$RP_1 + RR_1$	92.9		
			46	$RP_2 + RR_2$	94.2		
Simulink	2,719	782	13	RP_1	93.5	0.032	0
				RR_1	91.3		0
			23	RP_2	94.8	0.451	0
				RR_2	94.2		0
			26	$RP_1 + RR_1$	96.2		
			46	$RP_2 + RR_2$	97.7		
Scicos	10,053 (10 %)	7,137	13	RP_1	94.6	0	0.045
				RR_1	93.3		0
			23	RP_2	95.7	0.001	0
				RR_2	95.0		0
			26	$RP_1 + RR_1$	94.9		
			46	$RP_2 + RR_2$	96.2		

Table 4 Quadratic

Code	Total mutants	Equivalent mutants	Test data	Generation	Mutation score (%)	p value$_1$	p value$_2$
Manual	1,193	35	16	RP_1	84.4	0.991	0.020
				RR_1	84.3		0.023
			38	RP_2	90.4	0.360	0.005
				RR_2	89.4		0.004
			32	$RP_1 + RR_1$	88.5		
			76	$RP_2 + RR_2$	90.7		
Simulink	2,120	421	16	RP_1	83.5	0.301	0.027
				RR_1	81.8		0.001
			38	RP_2	85.4	0.663	0
				RR_2	85.4		0.012
			32	$RP_1 + RR_1$	85.8		
			76	$RP_2 + RR_2$	85.9		
Scicos	25,214 (10 %)	15,882	16	RP_1	83.8	0.053	0.003
				RR_1	82.8		0
			38	RP_2	84.6	0.620	0.042
				RR_2	84.3		0.014
			32	$RP_1 + RR_1$	84.5		
			76	$RP_2 + RR_2$	85.4		

For example, at the top of the table we have the program manually generated for the model, and a set of test cases of size 16 for the sets RP_1 and RR_1. Considering the mean of 30 RP_1 sets, a mutation score of 90.7 % was obtained. For RR_1, the mean obtained was 80.5 %.

5.4.2 Flow control

The results from the mutation testing are presented in Table 3. For the remaining mutants to be killed it was necessary to provide the 120 value to the high bound and 180 to the low bound, and to gather close values among the inputs.

5.4.3 Quadratic

The results from the mutation testing are presented in Table 4. To achieve a 100 % of mutation score it was necessary to satisfy the false condition of the switch block containing a threshold of 0.5, that required specific decimal values as inputs.

5.4.4 Duplex

The results from the mutation testing are presented in Table 5. In this model, inputs values close to each other were necessary to obtain a 100 % of the mutation score.

Table 5 Duplex

Code	Total mutants	Equivalent mutants	Test data	Generation	Mutation score (%)	p value$_1$	p value$_2$
Manual	4,416	184	11	RP_1	79.8	0.953	0
				RR_1	79.9		0
			23	RP_2	89.1	0.536	0
				RR_2	90.0		0
			22	$RP_1 + RR_1$	91.4		
			46	$RP_2 + RR_2$	94.8		
Simulink	5,944	2,873	11	RP_1	85.0	0.688	0
				RR_1	84.5		0
			23	RP_2	88.3	0.529	0
				RR_2	88.6		0
			22	$RP_1 + RR_1$	89.8		
			46	$RP_2 + RR_2$	90.7		
Scicos	17,718 (10 %)	12,969	11	RP_1	87.4	0.490	0
				RR_1	87.1		0
			23	RP_2	89.4	0.938	0
				RR_2	89.4		0
			22	$RP_1 + RR_1$	89.6		
			46	$RP_2 + RR_2$	92.2		

Table 6 Generic Voter

Code	Total mutants	Equivalent mutants	Test data	Generation	Mutation score (%)	p value$_1$	p value$_2$
Manual	12,566	706	16	RP_1	65.0	0.039	0
				RR_1	68.7		0
			108	RP_2	84.7	0	0
				RR_2	87.2		0
			32	$RP_1 + RR_1$	77.5		
			216	$RP_2 + RR_2$	89.7		
Simulink	13,267	2,619	16	RP_1	73.6	0.152	0.001
				RR_1	68.1		0
			108	RP_2	85.2	0.976	0.003
				RR_2	85.1		0.009
			32	$RP_1 + RR_1$	81.3		
			216	$RP_2 + RR_2$	90.2		
Scicos	46,718 (10 %)	32,936	16	RP_1	70.7	0.470	0
				RR_1	70.0		0
			108	RP_2	84.4	0.944	0
				RR_2	84.4		0
			32	$RP_1 + RR_1$	73.5		
			216	$RR_2 + RP_2$	88.0		

5.4.5 Generic voter

The results from the mutation testing are presented in Table 6. To complete the mutation score a specific value (3) was needed for the threshold, along with closer values among the other three inputs.

5.4.6 General considerations

Five case studies were used in order to verify the adequacy of test data generated by TeTooDS. Our focus is on the assessment of our generation strategies, nevertheless, without the support of TeTooDS as a testing tool, our experiments might had been unfeasible.

We had applied the mutation testing as a mean to evaluate our test data generation approaches. Since Simulink-like models are ultimately converted into low level code, we have considered the available mutant operators for the C language as adequate to our experimentation when using the code generated from our Simulink-like models.

The effect of a first possible approach is formalized into hypothesis as follows. **Null Hypothesis**, H_0: this hypothesis states that there is no real advantage in combining the inputs values using the pairwise approach. **Alternative Hypoth-**

esis, H_1: according to this hypothesis, the pairwise testing improves the ability of detecting faults in a model.

For each of the five models, six means were compared. In 19 of them, the mean of the generated sets using the pairwise combination was superior to the purely random generation. Despite this advantage, in only 10 of 30 comparisons we have obtained a statistical significance level higher than 95 %, resulting in an impossibility to refute the null hypothesis. Therefore, we can state that there is no evidence, in a statistical point of view, to indicate that the use of the pairwise technique is superior to the random generation for the considered models comparing RP_1 against RR_1 and RP_2 against RR_2.

Given this result, we decided to investigate whether even not reporting significant differences in the obtained mutation scores, the sets would be complementary or not. Thereby, we compared the sets to verify if the mutants that they are able to distinguish are the same or not. If different, we could use the pairwise generation along with the random one combined, in order to obtain a test set with greater effectiveness. This would be meaningless if the killed mutants are the same.

Thus, in each Table 2, 3, 4, 5, 6, we display the mutation scores obtained by joining together the test cases RP_i and RR_i. If the scores represented by RP_i and RR_i are greater than the values obtained by the sets RR_i and RP_i individually, this would indicate that the sets of mutants that they distinguish are, mostly, disjoint.

Clearly the mutation score obtained by the combined test sets is always greater than the score obtained by individual sets. Still, it is necessary to assess how significant this difference is, as when only increasing the number of test data (in RP_2 or RR_2) we could not achieve any improvements in the mutation scores. Therefore, we show in the the last column of the mentioned tables the level of significance when comparing the means of the individual sets and the combined ones.

For instance, the first line of the last column shows the p value considering the sets RP_1 against $RP_1 + RR_1$. In this case, all obtained values show a significance level greater than 95 %, making it possible for us to conclude that the use of randomly generated sets together with the ones generated by the pairwise approach may be the best alternative for the generation of test data according to our experiments.

One of our future goals is to define a set of techniques that can be used in an integrated way in embedded systems, based on the known concept that testing techniques are complementary and, therefore, should be used together. In this way, functional, structural and fault-based testing techniques may be used in an incremental strategy in which a test set created by one technique can be used as input to the next. In the case of test data generation, as in Zhan and Clark [48] for instance, an initial test set is required in order to apply the optimization algorithms. Our intention is to use an initial test set created using a more elaborated approach like the one suggested in this paper, i.e., combining random and pairwise techniques.

6 Conclusions and further work

In the development of embedded systems such as those embedded in many electronic devices, quality requirements are extremely valuable. Seeking to join these requirements with the need to reduce costs and delays in development, the industry identified in the model based development on a viable alternative.

TeTooDS provides a way to parse Simulink-like models and also helps on the selection of ranges of values for each input. It is possible to generate test data using a random or a pairwise approach, and a relation of the inputs with the outputs obtained can be displayed if a simulator is present.

Despite showing that we could not achieve a statistical significance to determine that the pairwise approach is superior to a random combination of values, we could show that using both approaches together it was possible to achieve better results and a statistical significance of at least 95 % in the conducted experiments.

As future work we plan to extend our strategy and to employ efforts in allowing the generation of values to cover decisions and conditions of a model. A strategy for the use of structural criteria in embedded system models needs to be defined, as well as instrumentation techniques. Furthermore, the incorporation of a fault-based criterion is desired. For this purpose, it is necessary to define or adapt mutation operators for this kind of model. One of the possibilities is to introduce errors by changing the operations performed by blocks.

A n-way combination of parameters and also a combination between different time steps may be useful in some cases and should be investigated in more detail. The testing tool underlying IPO strategy allows that the algorithm may be extended to such situations.

Finally, although there are several studies in this area, we believe that much can still be developed to contribute to the advancement of the state of the art and the state of the practice in the context of model based testing for embedded systems. This view is based mainly on the contact with engineers that work in the area and feel the lack of techniques and an integrated testing environment for these kinds of systems.

References

1. ADI (2012) Beacon tester. Available at http://www.adi.com/products_be_bss.htm
2. Alligood KT, Sauer TD, Yorke JA (2000) Chaos an introduction to dynamical systems. Springer, New York-Berlin
3. Araujo RF, Delamaro ME (2008) TeTooDS–Testing tool for dynamic systems. In: Tools session–Brazilian software engineering symposium, Brazil. SBC
4. Araujo RF, Durelli VS, Delamaro ME, Maldonado JC (2009) Test data generation from embedded system models: a systematic review. In: Brazilian Workshop on Systematic and Automated Software Testing, Brazil. SBC

5. Blackburn M, Busser R (1996) T-VEC: a tool for developing critical systems. In: Compass'96: Eleventh Annual Conference on Computer Assurance. National Institute of Standards and Technology, Gaithersburg
6. Blackburn M, Busser R, Nauman A (2004) Why model-based test automation is different and what you should know to get started. In: International Conference of Practical Software Quality and Testing, Software Productivity Consortioum, Washington, DC
7. Brillout A, He N, Mazzucchi M, Kroening D, Purandare M, Rümmer P, Weissenbacher G (2010) Mutation-based test case generation for simulink models. In: Proceedings of the 8th international conference on Formal methods for components and objects, FMCO'09, pp 208–227. Springer, Berlin
8. Broy M, Jonsson B, Katoen J, Leucker M, Pretschner A (eds) (2005) Model-based testing of reactive systems, advanced lectures, vol 3472 of lecture notes in computer science. Springer, Berlin
9. Campbell SL, Chancelier J-P, Nikoukhah R (2006) Modeling and simulation in Scilab. Springer, Scicos
10. Cavalcanti S, Papini M (2004) Preliminary model matching of the embraer 170 jet. J Aircraft 41(4):703–710
11. Chapoutot A, Martel M (2009) Abstract simulation: a static analysis of simulink models. In: ICESS '09: proceedings of the 2009 international conference on embedded software and systems, pp 83–92. IEEE Computer Society, Washington, DC
12. Cleaveland R, Smolka SA, Sims ST (2008) An instrumentation-based approach to controller model validation. pp 84–97
13. Cohen DM, Dalal SR, Fredman ML, Patton GC (1997) The AETG system: an approach to testing based on combinatiorial design. IEEE Trans Softw Eng 23(7):437–444
14. Delamaro ME, Maldonado JC (1996) Proteum–a tool for the assessment of test adequacy for c programs. In: Proceedings of the conference on performability in computing systems (PCS 96), pp 79–95
15. DeMillo RA, Lipton RJ, Sayward FG (1978) Hints on test data selection help for the practicing programmer. IEEE Comput 11(4):34–41
16. Embraer (2012) Empresa brasileira de aeronáutica s.a. Available at http://www.embraer.com
17. Gadkari AA, Yeolekar A, Suresh J, Ramesh S, Mohalik S, Shashidhar KC (2008) Automotgen: automatic model oriented test generator for embedded control systems. In: CAV '08: proceedings of the 20th international conference on computer aided verification, pp 204–208. Springer, Berlin
18. Geensys (2012) Safety test builder. Available at http://www.geensys.com/?Outils/SafetyTestBuilder
19. Grindal M, Offutt J, Andler SF (2005) Combination testing strategies—a survey. Softw Test Verif Reliab 15(3):167–199
20. Hall M (1986) Combinatorial theory. Wiley-Interscience Series in Discrete Mathematics. Wiley, Hoboken
21. Henry JE (2000) Test case selection for simulations in the maintenance of real-time systems. J Softw Maint 12(4):229–248
22. Henry JE, Stiff JC, Shirar AJ (2003) Assessing and improving testing of real-time software using simulation. In: ANSS '03: proceedings of the 36th annual symposium on Simulation, pp 266–274. IEEE Computer Society, Washington, DC
23. Hsiung P-A, Chen Y-R, Lin Y-H (2007) Model checking safety-critical systems using safecharts. IEEE Trans Comput 56(5):692–705
24. INRIA Rocquencourt (2012) Scicos. Available at http://www.scicos.org
25. Korn GA (2007) Advanced dynamic-system simulation: model-replication techniques and Monte Carlo simulation. Wiley-Interscience, New York
26. Kuhn D, Wallace D, Gallo AMJ (2004) Software fault interactions and implications for software testing. IEEE Trans Softw Eng 30(6):418–421
27. Lei Y, Kacker R, Kuhn DR, Okun V, Lawrence J (2007) Ipog: a general strategy for t-way software testing. In: ECBS '07: Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, pp 549–556. IEEE Computer Society, Washington, DC
28. Lei Y, Tai K (1998) In-parameter-order: a test generation strategy for pair-wise testing. In: Third IEEE high assurance systems engineering symposium, pp 254–261. IEEE Computer Society Press
29. Lott C, Jain A, Dalal S (2005) Modeling requirements for combinatorial software testing. In: A-MOST '05: proceedings of the first international workshop on Advances in model-based testing, pp 1–7. ACM Press, New York, NY
30. Marwedel P (2006) Embedded system design. Springer, Secaucus, NJ
31. Mathur A (2008) Foundations of software testing. Pearson Education
32. Meenakshi B, Bhatnagar A, Roy S (2006) Tool for translating Simulink models into input language of a model checker. In: ICFEM, pp 606–620
33. Myers GJ, Sandler C, Badgett T, Thomas TM (2004) The art of software testing, 2nd edn. Wiley, New York, NY
34. Perry W (2006) Effective methods for software testing, 3rd edn. Wiley, New York, NY
35. Pressman RS (2006) Software engineering. McGraw Hill, New York
36. Proakis JG, Salehi M, Bauch G (2004) Contemporary communication systems using MATLAB and Simulink, 2nd edn. Thomson-Brooks/Cole, Belmont, CA
37. Reactive Systems (2012) Reactis. Available at http://www.reactive-systems.com
38. RTCA (1993) Software consideratons in airborne systems and equipment certification. Technical Report DO-178b/ED-12B, RTCA Aviation Standards, Inc
39. Satpathy M, Yeolekar A, Ramesh S (2008) Randomized directed testing (redirect) for simulink/stateflow models. In: EMSOFT '08: Proceedings of the 8th ACM international conference on Embedded software, pp 217–226. ACM, New York, NY
40. Schroeder P, Bolaki P, Gopu V (2004) Comparing the fault detection effectiveness of n-way and random test suites. In: International Symposium on Empirical Software Engineering, 2004. Proceedings. 2004 ISESE '04. pp 49–59
41. Sims S, DuVarney DC (2007) Experience report: the reactis validation tool. SIGPLAN Not 42(9):137–140
42. T-VEC (2012) T-vec. Available at http://www.t-vec.com
43. The MathWorks Inc (1999) Simulink: dynamic system simulation for MATLAB. Prentice-Hall, Englewood Cliffs, NJ
44. The Mathworks Inc (2012) Matlab and simulink for technical computing. Available at http://www.mathworks.com
45. Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A (2000) Experimentation in software engineering. Kluwer, Dordrecht
46. Zhan Y, Clark J (2004) Search based automatic test-data generation at an architectural level. In: Genetic and Evolutionary Computation–GECCO-2004, Part II, vol 3103 of Lecture Notes in Computer Science, pp 1413–1424. Springer, Seattle
47. Zhan Y, Clark J (2005) Search-based mutation testing for simulink models. In: GECCO 2005: proceedings of the 2005 conference on Genetic and evolutionary computation, vol 1, pp 1061–1068. ACM Press, Washington, DC
48. Zhan Y, Clark JA (2008) A search-based framework for automatic testing of matlab/simulink models. J Syst Softw 81(2):262–285