Journal of Computer Science

Original Research Paper

Evaluation of Test Case Generation based on a Software Product Line for Model Transformation

[1]Alexandre Augusto Giron, [2]Itana Maria de Souza Gimenes and [2]Edson OliveiraJr

[1]*Federal University of Technology, Parana, Toledo-PR, Brazil*
[2]*Department of Informatics, State University of Maringá, Maringá-PR, Brazil*

Article history
Received: 31-10-2017
Revised: 17-11-2017
Accepted: 18-1-2018

Corresponding Author:
Alexandre A. Giron
Federal University of Technology, Parana, Toledo-PR, Brazil
Email: alexandregiron@utfpr.edu.br

Abstract: Model-Driven Engineering (MDE) supports model evolution and refinement by means of model transformations at several abstraction levels. Validating these transformations is essential to ensure the quality and correctness of such models. However, MDE transformations become more complex to validate, for example, when they are implemented in different languages. One particular example is the transformation of the SyMPLES approach. SyMPLES is a development approach for embedded systems, which is based on concepts of both Software Product Lines (SPL) and MDE. SyMPLES has a model transformation process which creates Simulink models from SysML models. This paper presents a case study which applies test case generation based on SPL to validate this model transformation. An SPL was used to generate a set of test cases based on coverage criteria. The results showed that the test cases generated uncovered errors in the transformation of SyMPLES. In addition, a comparison with the test case generation based on metamodel is presented, in order to analyze the effectiveness of the techniques. The coverage criteria made it possible to reduce the number of test cases generated, thus minimizing test effort and time.

Keywords: MDE Validation, Software Product Line, Embedded Systems

Introduction

An embedded system is a computer system incorporated in a larger product (Marvedel, 2003). The development process of these systems is different from traditional systems due to their specific nonfunctional requirements, such as, power consumption, performance, hardware and software integration, real time and cost. Therefore, specific approaches are needed to support their development process.

Model-Driven Engineering (MDE) and Software Product Lines (SPL) are complementary approaches that contribute to improving the embedded systems development process. MDE allows the generation of applications and models by means of model transformations at distinct abstraction levels (Mellor, 2004) and SPL supports non-opportunistic reuse of common artifacts from the same domain to customize new applications (Van der Linden *et al*., 2007). One of the goals of MDE is to facilitate the code generation from specification models, which can be carried out automatically using MDE transformations.

The SysML-based Product Line approach for Embedded Systems (SyMPLES) (Silva *et al*., 2013; Fragal *et al*., 2013) is an approach to support the embedded systems development process, which combines both MDE and SPL concepts. SyMPLES is composed of activities for both variability management and model transformation. It guides the development using annotations in SysML models (Friedenthal *et al*., 2009) to specify the system. The annotations support the variability resolution to configure specific products.

SyMPLES transforms configured SysML models into Simulink models. SysML is a modeling language for the specification of dynamic systems, at a high abstraction level, whereas MATLAB/Simulink (Mathworks, 2017) is an environment which supports modeling, system simulation and C/C++ code generation.

Validation of MDE transformations is important to ensure quality (Fleurey *et al*., 2004). If the models are

Science Publications

© 2018 Alexandre Augusto Giron, Itana Maria de Souza Gimenes and Edson OliveiraJr. This open access article is distributed under a Creative Commons Attribution (CC-BY) 3.0 license.

derived automatically by the MDE transformations, then their quality will depend on the correctness of the transformation (Küster and Abd-El-Razik, 2006).

Software testing techniques have been used to validate MDE transformations (Küster and Abd-El-Razik, 2006), but these techniques are significantly different from testing traditional software. This is due both to the declarative nature of some transformation languages and to the complex structure of the specification models, which can have different types of elements and arrangements between them (Tiso *et al*., 2012). Furthermore, model transformations can be implemented by different languages when divided into smaller transformation steps, increasing their complexity. Thus, validation becomes more difficult as the testing must take into consideration all the transformation steps and the transformed models (output models).

Analyzing the model transformation of the SyMPLES approach, a need of validation was identified, because:

- It was evaluated in just one application example, developed to specify a system to Yapa 2 board, responsible for the flight control of an UAV (Fragal *et al*., 2013), in the context of the Brazilian National Institute of Science and Technology for Critical Embedded Systems (INCT-SEC)
- Any MDE transformations must be validated to analyze whether its execution produces expected results
- SyMPLES model transformation structure was designed with two transformation steps, each one designed in a different language, using inputs and producing output models. This "chain-based" design could complicate the validation of the model transformation as a whole
- The input domain (SysML metamodel) for the model transformation can produce a big set of test cases. It is important to seek alternatives to reduce the size of the test cases, for example using coverage criteria and generation policies, in the context of MDE transformations (Fleurey *et al*., 2004; Küster and Abd-El-Razik, 2006)

This paper presents a case study for the validation of the SysML to Simulink model transformation. In synthesis, the case study involves an MDE validation technique using a SPL as input to the test case generation, in the context of the embedded systems development. The main objective of this paper is to evaluate the test case generation using SPL to the validation of the model transformation of the SyMPLES approach.

In a previous published work (Giron *et al*., 2017), the model transformation of the SyMPLES was tested using test case generation from the SysML metamodel. Now, another objective in this study is to compare the results with the previous publication. The same model transformation is used to compare the techniques of test case generation: From the SysML metamodel (previous publication) and the generation from a SPL (this paper).

This paper is organized as follows: Related Work is presented in the second section; detailed information of the SyMPLES approach and its MDE transformation are presented in the third section. The fourth section presents concepts on MDE validation; the test case generation technique is presented in the fifth section; the sixth section presents the application of the technique and its results; and conclusions, contributions and future work are presented in the seventh section.

Related Work

Validation of MDE transformations was investigated in many studies. There is a variety of techniques for validation as Model-Based Testing, formal verification and validation based on common software testing techniques.

Brottier *et al*. (2006) presented a test case generation process based on metamodel. This process consists on creating partitions of the metamodel, using equivalence classes, aiming to reduce the input domain. These partitions are used to create model fragments that can be used to generate input models for testing. A limitation found is that the scope is reduced to the test case generation. Our work focuses on the test case generation but other activities are also considered (i.e., Run tests activity).

Tiso *et al*. (2012) provide a development method for MDE transformations and two approaches for testing: Static test and dynamic test. However, no technique for test case generation is discussed or proposed, thus it is assumed the tester already has the input models to the test execution.

Another approach for validation of transformations is the Model-Based Testing (Guerra, 2012; Lano *et al*., 2015). Differently from common software testing, this kind of testing is usually performed with a symbolic execution of the models. These models are used to specify the desired behavior of the transformation, for example, state machine models.

Lin *et al*. (2005) proposed a framework and a tool for transformation testing. The tool allows the mapping verification between input/output models and shows the visualization of the differences between these models.

In this study, validation is based on functional testing, mostly because the SysML to Simulink model transformation implementation is composed of two different languages. SyMPLES does not provide any formal model to perform model-based testing of the transformation and this kind of testing is out of the scope of this paper. The main three activities of testing MDE transformations were considered in this

study, from test case generation to the execution and analysis. In addition, a comparison with previous work is performed in this study.

Test Case Generation based on the SysML Metamodel

A previous work tackled the validation concern of the SyMPLES model transformation using a metamodel-based test case generation technique (Giron *et al*., 2017).

The previous work technique was applied using the SysML metamodel, with two generation policies and a set of coverage criteria. The generation policies were used to define the size of a test case, because the bigger a test case is, the more difficult to interpret it (Sen *et al*., 2009). In addition, a bigger set of test cases increase the test effort and time, therefore, an automated environment, an analysis of the SysML metamodel and coverage criteria were used.

The generation policies were applied to define how many elements compose a SysML model. Each SysML model was used to test the model transformation of the SyMPLES approach. With regard to the policies, they are described as follows:

- **(N to 1) policy**: A limited set of SysML elements is inserted in the same diagram. Heuristics can be applied to define the N value. For example, each test case could group at most five of the possibilities from one specific type of relationship between two elements
- **(1 to 1) policy**: One diagram for each new element. Using this policy it is easier to find out the cause of an error when it occurs, but it could increase the size of the set of test cases

With regard to the input domain for the test case generation, the coverage criteria determined how much of the SysML metamodel was used. Also, in order to reduce the test case set, a strategy applied was to generate only elements that could be used by the transformation rules. For example, certain diagrams from the SysML language were not took into account in the SyMPLES model transformation. Therefore, these diagrams and elements were excluded. The result of these strategies was a reduction of the number of test cases and avoided to generate useless test cases.

After running tests using the test cases generated and comparing the policies used, the effectiveness was analyzed. The effectiveness was calculated in terms of how many test cases were able to identify an error in the model transformation. Comparing the policies used, the effectiveness was about 18% (1 to 1) and 22% (N to 1).

A characteristic of the test case generation based on metamodel is the set of test cases are generic. Each test case is composed of a SysML model (and the expected result of the transformation), but this model is generic. This means it does not represent a real system specification, for example. This is not considered to be a problem, however, if the test could be more specific to the model transformation, it would improve the effectiveness of the test case set.

The SyMPLES Approach

SyMPLES can be divided into two parts: Domain engineering, which contains the SPL related activities, as variability management; and application engineering, composed of a model transformation.

SPL in SyMPLES is specified with SysML models with stereotypes to variability management and for functional blocks. The transformation of SyMPLES can be applied on the configured SysML models to obtain corresponding Simulink models.

SyMPLES uses profiling mechanism for creating two extension profiles of the SysML language, as follows (Silva *et al*., 2013):

- The SyMPLES Profile for Functional Blocks (SyMPLES-ProfileFB) specifies the types of functional blocks by means of a set of stereotypes. These stereotypes provide additional semantics to the SysML blocks. Therefore, this profile helps to map SysML blocks to Simulink blocks
- The SyMPLES Profile for Representation of Variability (SyMPLES-ProfileVar): Represents the variabilities of an SPL from a set of stereotypes and aggregate values to the elements of the SysML diagrams. Each product of the SPL is a SysML configured model

In this approach two processes are defined to the SPL activities, as follows:

- SyMPLES Process for Product Lines (SyMPLES-ProcessPL) consists of activities related to SPL artifacts creation
- SyMPLES Process for Identification of Variabilities (SyMPLES-ProcessVar) is based in the SMarty approach (OliveiraJr *et al*., 2010). It aims to support the SPL variability management, from variability identification to product configuration

The functional blocks profile was created to support the model transformation in SyMPLES approach. This model transformation maps SysML elements to Simulink blocks. Each SysML model can be transformed to a Simulink model, allowing, for example, the simulation of embedded system specified. Therefore, Simulink models generated from this transformation are closer to

the system implementation. In addition, Simulink models can be used for code generation, provided by the MATLAB/Simulink tool.

SysML to Simulink Model Transformation

The model transformation of the SyMPLES approach is divided into three steps: (i) Configure a SysML model, (ii) run an ATL transformation (Jouault and Kurtev, 2005) and (iii) generate functional blocks, as shown in Fig. 1.

Configure a SysML Model

In this step, a SysML model with SyMPLES stereotypes must be configured. The model can be composed of four diagrams: Block Definition, Internal Block, State Machine and Parametric. The root diagram is the Block Definition, used to describe the main blocks of the system. The Internal Block represents the internal relationship of a block based on block instances, therefore one Internal Block diagram can be used for each main block specified in the Block Definition. The behavior can be specified in the State Machine diagram and the Parametric diagram specifies block constraints, values and properties. If a block will be used in the system, or if it is optional, then these variabilities must be resolved to configure the SysML model.

The variabilities are specified in a Block Definition Diagram using variation points, defined in the SyMPLES approach. An example of the definition of an SPL for the Mini-UAV, used in this study, is presented in Fig. 2.

In Fig. 2 three variation points are defined: Barometer, Servos and Camera. SyMPLES defines the variation point stereotype which means that it must be one Internal Block Diagram to specify such variability.

The SysML model with SyMPLES stereotypes can be imported to the pure::variants tool (Beuche, 2012), which creates a Variant Model Descriptor (VDM). In Fig. 3 an example of a configured VDM is shown. The Feature Model in Fig. 3 shows three variabilities: Two options mutually exclusive for the Barometer sensors; two options mutually exclusive for the Camera; and two options for the Servos. The variabilities are resolved in this model, then, they are automatically reflected in an output SysML model.

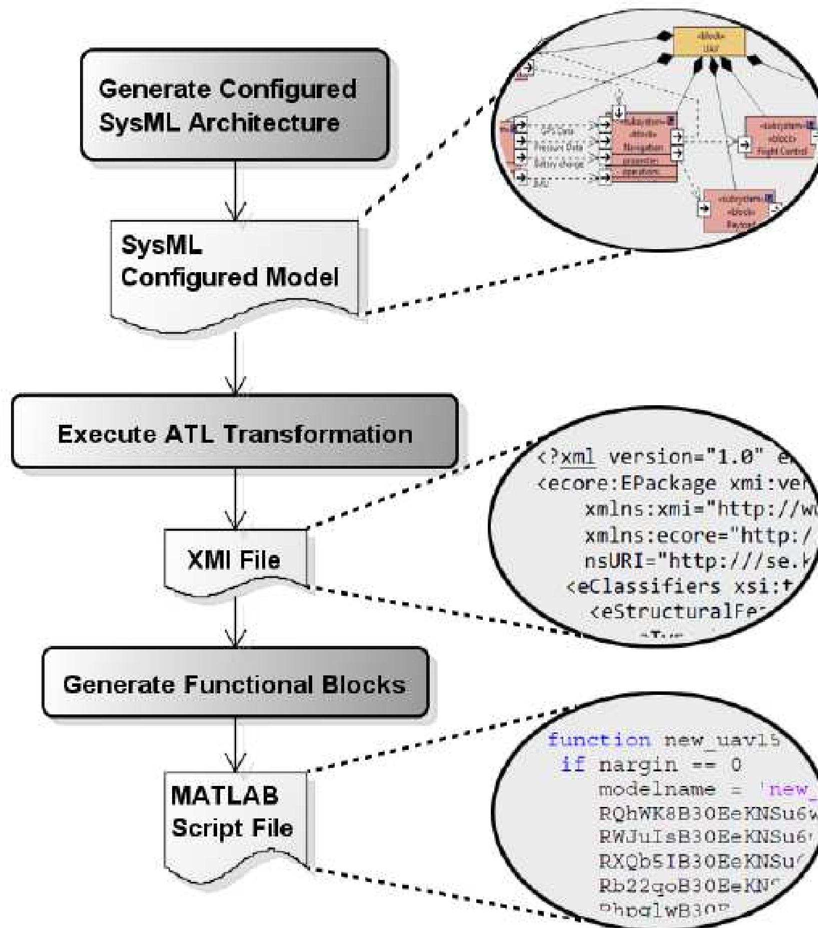

Fig. 1: SysML to Simulink model transformation, provided by SyMPLES approach

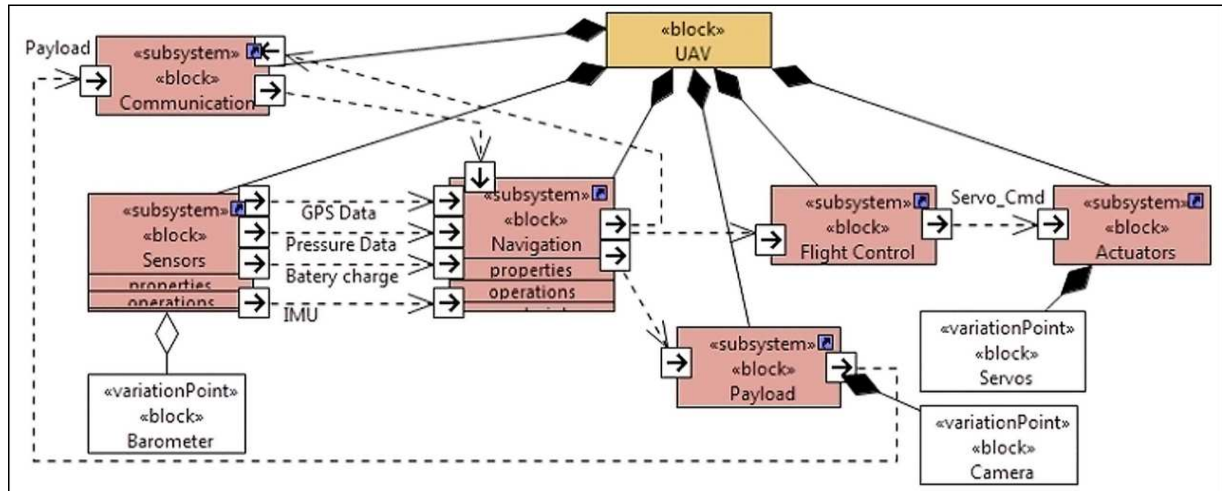

Fig. 2: Block Definition Diagram of the Mini-UAV (Fragal *et al*., 2013)

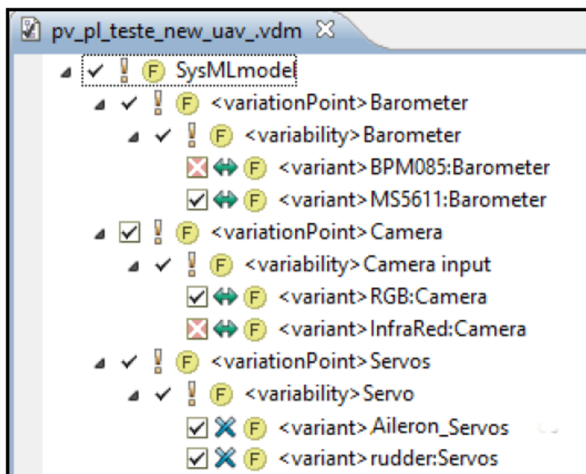

Fig. 3: VDM Model representing the features of the Mini-UAV SPL, generated in pure::variants tool

Execute ATL Transformation

This step uses ATL rules to select relevant information about the SysML model, like element attributes, stereotypes and graphic data (i.e., element position and size). Each element with relevant information is stored in one XMI intermediary model. This intermediate transformation using an XMI intermediary model makes this process more flexible to deal with EMF-based editors.

Generate Functional Blocks

This step is implemented in Java and uses the XMI model produced in the previous step and the main file of the SysML model, known as UML file. The UML file must be used because it contains values that are referenced by the XMI model and to retrieve the SyMPLES stereotypes in the SysML model. In this step, a MATLAB script is generated and it represents the Simulink model. With its execution, the Simulink model can be visualized.

The SyMPLES transformation process can be classified as a Model-to-Model (M2M) transformation, receiving as input models the SysML models with SyMPLES stereotypes. The SysML configured model is used as input to the transformation, producing an intermediary file (XMI), which is based on the Simulink metamodel (Biehl *et al*., 2010). The XMI file is an intermediate model. The final Simulink model is generated only after the "Generate Functional Blocks" step.

SyMPLES Transformation Example

An example of input model can be visualized in Fig. 4. It is composed of a SysML Internal Block Diagram and its Block Definition diagram is the same as in Fig. 2. The Navigation block from the Block Definition Diagram is composed of a block "Yapa2", which has the subsystem stereotype, defined in SyMPLES. This means that a Simulink subsystem block will be generated after the transformation.

The obtained result after the execution of the transformation is an output model composed of a MATLAB script, known as M-File. Then, this script can be executed in MATLAB to produce the Simulink model. Figure 5 shows the Simulink model.

The SyMPLES transformation can be complex to validate because its implementation is divided into two steps, each one designed in a different language. The first one used ATL language and the functional block generation was written in Java. In addition, the transformation uses the SysML and Simulink metamodels and reads stereotypes information from SyMPLES profiles when it transforms the input models. Therefore, it is important to take into consideration all of these concerns for the validation process of the SyMPLES transformation.

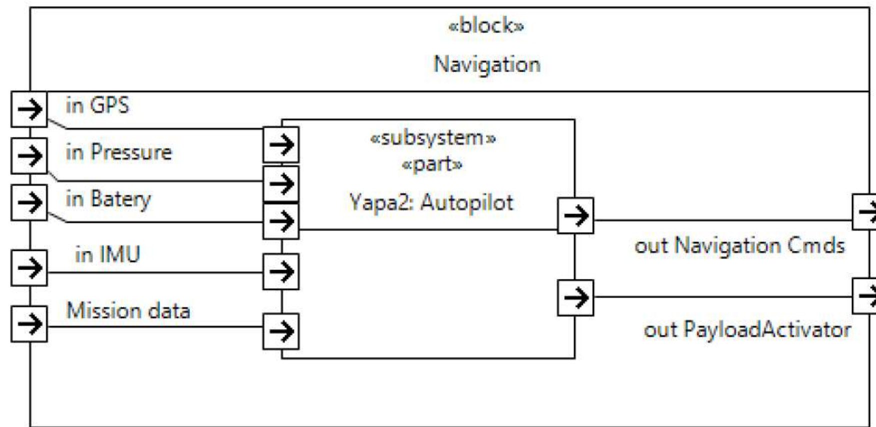

Fig. 4: Example of input model to the transformation of the SyMPLES approach. Adapted from Fragal *et al*. (2013)

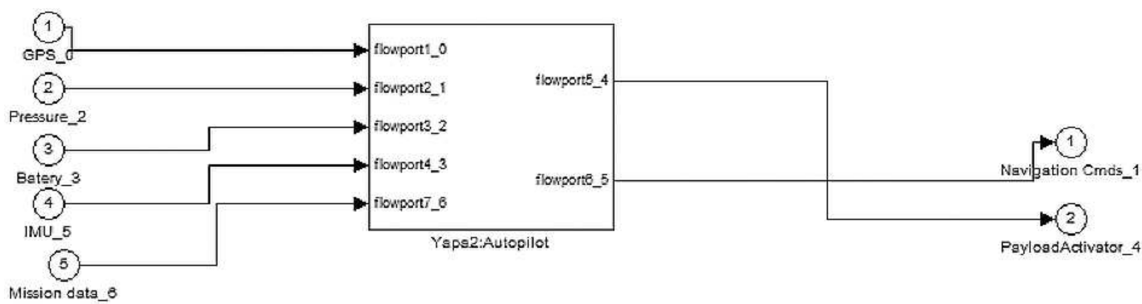

Fig. 5: Example of Simulink model generated by the transformation. Adapted from Fragal *et al*. (2013)

In the previous work, a first attempt to validate the model transformation of SyMPLES was performed. The test case generation technique was based on the SysML metamodel. The test cases generated were generic because the only information used was the SysML metamodel and SyMPLES stereotypes.

In this study, an alternative technique to test case generation is proposed. As the SyMPLES model transformation transforms models from SPL specifications, the main idea is to use an SPL to provide test cases for the model transformation. The details of the proposal evaluated in this case study are explained in the next section.

Validation of MDE Transformations

Approaches based on software testing are frequently used in industry and can be applied to validate MDE transformations (Fleurey *et al*., 2004). Two main types of testing can be applied: Black-box testing, or functional testing, in which the input models are compared to the output models after the transformation; and White-Box testing, or structural testing, in which the internal aspects of the model transformation are analyzed.

As previously mentioned, a transformation can be written in different languages (e.g., ATL, QVT) and even common programming languages. In addition, transformations become more difficult to test when the same transformation is designed into steps, where each one can also be written in different languages, like the transformation of the SyMPLES approach. In these cases, a functional approach is more appropriate.

Regardless the testing type, to validate model transformations at least three steps must be followed (Küster and Abd-El-Razik, 2006): Test Case Generation, in which the test case are generated accordingly to a coverage criterion; Oracle definition, which defines the expected result of a test; and Test Execution, to determine and analyze the testing results.

In the Test Case Generation, two factors must be taken into account: The size of one test case and the size of the set of test cases. The size of one test case is the number of elements of the model. A test case with few elements facilitates both its comprehension and an efficient diagnostic when an error is found. However, decreasing the size of the test case may result in an increase of the set of test cases. When the size of the set of test cases is too big, the test becomes unfeasible (Fawaz *et al*., 2015). Therefore, reducing the size of the set is important to reduce the test time by using coverage criteria and generation policies (Fleurey *et al*., 2004; Küster and Abd-El-Razik, 2006).

DOI: 10.3844/jcssp.2018.108.121

Considering the Test Execution, Tiso *et al*. (2012) define two approaches to transformation testing: Checking static properties (static testing) and analyzing the execution (dynamic testing). Static testing refers to verifying properties in the output models, like checking whether certain attributes are present in the output model. On the other hand, dynamic testing refers to analyze the execution of the output model, if it is executable.

The result analysis process from the execution of the tests in model transformations is also important. A transformation is usually based on rules that map elements from input models to the corresponding one in the output models. Thus, a wrong use of the transformation rules may lead to errors, classified as (Küster and Abd-El-Razik, 2006):

- **(Type 1) Metamodel coverage**: The transformation rules have been implemented, but they are not sufficient to map all elements that the metamodel possess. An example is when the rules can only be applied to certain kinds of elements, thus the other kinds of elements are not mapped
- **(Type 2) Syntactically incorrect models**: When the transformation rule cause generation of an output model that does not comply with the output metamodel
- **(Type 3) Semantically incorrect models**: When the transformation rules are applied to an input model and the output model is produced syntactically correct, but it does not produce a model with the expected elements. For example, when an input model with elements is transformed but some elements are missing in the output model. Therefore, the output model is not a correct transformation from the input model
- **(Type 4) Ambiguity**: The same transformation rule produces different results from the same input model
- **(Type 5) Errors due to incorrect coding**: Included here all the other types of common errors and the codification errors. Examples are the incorrect primitive types (integer, floating point) and memory references out of bounds

Test Case Generation Based on SPL Modeled with SyMPLES

This section presents the test case generation based on SPL. The SPL must be modeled with SyMPLES in order to generate test cases for the model transformation of the SyMPLES.

Due to the fact that SyMPLES has variability management, the possibility of generating test cases from the SPL was identified. Each product from a SPL is a SysML model with resolved variabilities, so it can be used as an input model to test the transformation.

As the model transformation of the SyMPLES needs a product from a SPL (i.e., a model) to transform it into a Simulink model, a SPL can be used as the “input domain” to generate several products to test if the transformation can produce the expected results (i.e., the Simulink model).

It is worthy to note that, in this case study, the focus is to test the SyMPLES model transformation, **not the SPL**. This paper will not perform any SPL testing techniques. A premise used in this case study is that the SPL must be modeled with SyMPLES approach and the software components of the SPL are already tested. An analogy that can be made is that the model transformation corresponds to the System Under Test (SUT) and the SPL corresponds to the input data domain of the SUT. Obviously, it is a reduced domain as it discards elements that are not used by the transformation. The SPL provides a family of SysML models as input to test the model transformation.

However, the test case generation using SPL for model transformations has little research available. Possibly, this is due to the fact that this is a particular situation: An MDE transformation with specific characteristics needs validation and the proposal is to use an SPL as a source of test cases, not to perform SPL testing, but to test the MDE transformation in the context of the Embedded Systems and of the SyMPLES approach.

According to Lochau *et al*. (2012), SPL testing has an issue about the number of products that can be generated from the SPL. This concept is the same when using an SPL as input domain to test the model transformation. If the SPL has too many products then the test could be difficult to manage. To alleviate this problem, a coverage criterion can be applied to the SPL to determine the maximum number of generated models for testing and therefore allowing partitioning the SPL. Other approaches can be used as well, for instance optimization algorithms (Fleurey *et al*., 2004) or search-based software testing (Anand *et al*., 2013).

Therefore, to apply the test case generation based on SPL, to test the model transformation of SyMPLES, three definitions are made, as follows:

- SPL must be modeled using SyMPLES. This is a requirement to the transformation itself, otherwise it will not produce any output model
- Domain definition: The SPL is used to provide the test cases, therefore it is needed to perform a Feature Analysis in the SPL in order to calculate the maximum amount of products available in the SPL. This number reflects the size of the set of test cases
- Coverage criteria: It is also needed to define the coverage on the test case set, in terms of the quantity of test cases. It is important to highlight that, if increasing the size of the test case set, then it could increase the test effort and time

The main idea of this technique is to generate specific test cases, related to the SPL used, because this model transformation requires SyMPLES stereotypes. Basically, in SyMPLES the specification of an SPL of a system will have blocks with stereotypes and its variabilities. The stereotypes define which Simulink block will be generated at the output model level. Therefore, using a real world SPL would be possible to generate specific test cases, to test at least the elements used and allowed to be transformed. Then, a test case will be created based on one product of the SPL. The calculation of the amount of test cases for the generation can be performed with the SPL Feature Analysis procedures.

SPL Feature Analysis

The Feature Analysis can be performed based on the Binary Decision Diagram (BDD). Its implementation is based on the SPLOT tool, in this study. Comparative studies in feature analysis showed that BDD is efficient in terms of execution time (Mendonca *et al.*, 2009; Benavides *et al.*, 2007).

BDD uses a logic structure to represent a boolean function, composed of decision nodes and terminal nodes (0 and 1). Each node represents a boolean variable and all the paths will lead to the boolean value 1 when the function is true. Connections will lead to left or right and correspond to the value 0 and 1, respectively. Figure 6 presents an example of a BDD and its corresponding Truth Table of a boolean function S. The function can be resolved using the values in the table to traverse the path in the BDD.

The feature model of an SPL can be represented with a BDD. In an example, in an SPL with two mutually exclusive options to configure, the possible configurations can be defined equivalently using a BDD with an XOR function. Options for SPL features in SyMPLES are *alternative_OR*, *alternative_XOR*, *optional* and *mandatory*. In summary, SPL constraints are mapped to boolean functions in a BDD.

Implementation Details

The SPL-based test case generation was implemented in this case study using the Java language and an SPL specified using SysML with stereotypes of the SyMPLES approach. The test case generator allows two types of coverage criteria: Partial and total. They are based on the percentage of the maximum amount of products from the SPL, calculated using Feature Analysis previously presented.

Figure 7 shows the process of SPL-based test case generation and the Test Execution. Initially, the SPL definition is taken as input. The SPL definition includes models specified according to SyMPLES and an XML specification. They are used to generate VDM models, allowing the configuration of each product according to the coverage criterion. In addition, the specification in XML format is used because it is compatible with the BDD feature analysis (based on SPLOT tool as mentioned before).

A limitation of this implementation is that the VDM configuration must be manual, in the test case generation. Other SPL can be used for test case generation with the tool implemented in this case study, but it must be specified according to SyMPLES, otherwise it would need a tool to automate the generation of the products. The implementation created in this study is specific to the SyMPLES approach, but the concept would be the same for other types of SPLs.

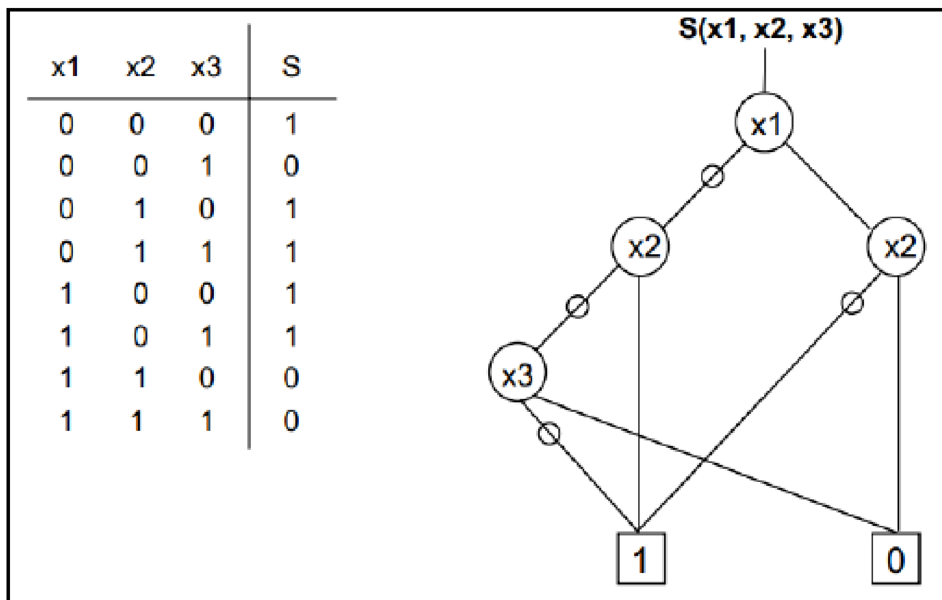

Fig. 6: BDD example

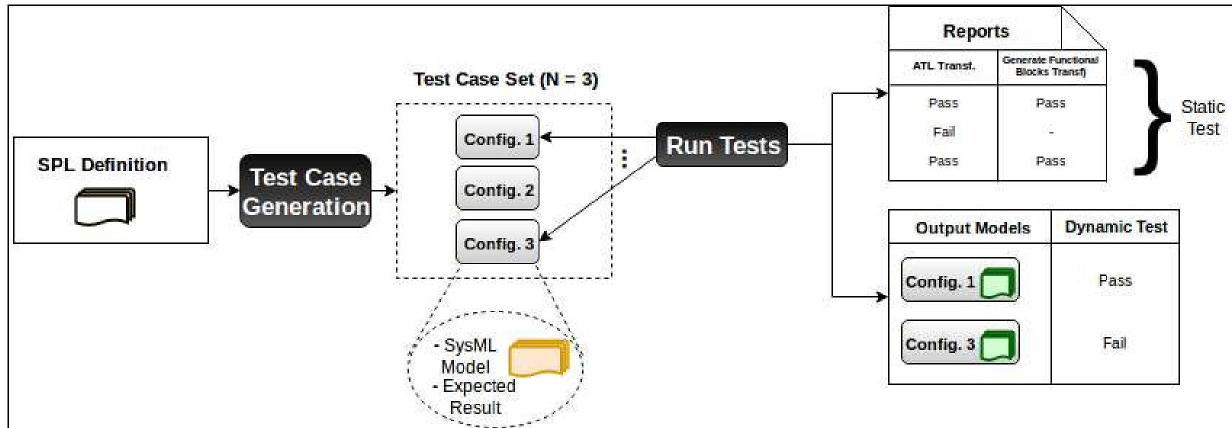

Fig. 7: SPL-based test case generation and run tests activity

After the VDM generation, feature configuration must be performed, then a script must be executed to create the SysML models, according to the configured VDM. Such script is generated jointly with the VDM. Then, the SysML models generated by the script and corresponding to each VDM can be used as input to the transformation test. Each SysML model represents a configuration of the SPL (i.e., Config. 1 in Fig. 7). This configuration also the expected results of the transformation, but this is not part of the test case generator tool.

In this case study, a table was used to map the input and output elements, using the transformation rules extracted from the SyMPLES documentation. For example, if an input model contains a determined element, in the output model a corresponding element (or elements) must be present. Therefore, it is possible to check if the transformation test has passed or not.

It is worthy to highlight that if an error in the transformation is identified (for example, in the ATL step of the model transformation), then the second step cannot be tested because there is no output model from this step. This scenario is showed in the reports of Fig. 7. If an error is detected in the Run Tests activity, the transformation will not generate the output model. This scenario is not a problem, because the main objective of a test case is to find an error in the transformation.

Static Test

The static test is basically composed of the execution of model transformation using test cases generated and analyzing its results. The execution of the tests aims to run the transformation in a suitable environment. Then, one or more intermediary models or output models can be produced, but only if the transformation executes as expected, without any execution errors.

It is important to highlight using proper tools to automate the execution tends to decrease the test time and facilitate the result analysis. The test case generator implemented in this case study also generates an ANT script to run the tests automatically.

Dynamic Test

Dynamic test can be applied if the output models produced by the transformation are buildable or executable (Tiso *et al*., 2012). If the output models do not execute correctly, then the transformation was not able to generate them as expected. Therefore, the dynamic test was included to provide a higher level of validation.

To automate the dynamic test, a MATLAB script was used, developed in previous work. The script executes each output model using the *feval* command. If an error occurs, it is stored in a text file with prefix *logDynamicTest*. The following information composes the log file: Error number (or count), the name of the output model, error message and the line number. This information is needed to help to find the cause of the error in the model. Finding the cause of an error often requires knowledge of its implementation.

Validation of the SysML to Simulink Model Transformation

In this section, the case study scenario is presented. The test case generation based on SPL was applied to the model transformation of SyMPLES.

Hypothesis of this Case Study

In this case study, the hypothesis is described as follows:

- The test of the model transformation of SyMPLES using an SPL modeled with SyMPLES is more specialized to find errors compared to a metamodel-based technique (of test case generation)

DOI: 10.3844/jcssp.2018.108.121

In other words, an SPL-based technique would allow finding more errors in the transformation with fewer test cases generated, when compared with a metamodel-based technique. The activities performed in this case study are described as follows.

Activities Performed

Three main activities were performed to validate the transformation of the SyMPLES approach: Test Case Generation, Test Execution (Run Tests) and Result Analysis.

As previously mentioned, for the validation of the SyMPLES transformation the Test Case Generation by SPL was performed. The SPL used as input to the test case generation was the Mini-UAV specification presented in Fig. 2 and 3. An analysis is shown in Fig. 8 related to the variability "Barometer" and "Servos". The first variability shows two mutually exclusive options: BPM085 and MS5611, mapped to the BDD as a XOR function. The "Camera" variability is also mapped as a XOR function and the "Servos" variability is mapped as an OR function. In Fig. 8, there are two paths that lead to 1-terminal to the XOR structure and three to the OR structure.

The result of the SPL analysis and considering all of the variabilities is a maximum amount of twelve possible configurations: Two for the Barometer, combined with two for the Camera and three for the Servos, as shown in Table 1. Therefore, twelve SysML models can be generated as products to test the model transformation.

Table 2 shows the results from generating test cases using this Mini-UAV SPL. Using one coverage criteria will define how much SPL products (in this case SysML models) will be tested in the transformation. The last column of Table 2 shows the quantity of errors found when the first step (Step 1: ATL transformation) of the model transformation was tested.

Twelve products were generated using the total coverage criterion. Each product is a SysML model composed of one Block Definition, Internal Block and State Machine Diagrams and then used to test the transformation. Then, each model associated with the expected result of the transformation composes the test case.

It is worthy to note that the Run tests activity was divided into two steps. This is due to the fact that the transformation is structured in two steps: ATL transformation (Step 1) and the Generation of Functional blocks, written in the Java language (Step 2). Table 3 presents an example report, related to the results of the test of Step 1, related to the total coverage criteria. The test cases were named variant because each one is a variant configuration of the SPL.

Table 1: Total amount of possible configurations of the SPL

Variability	Function	Possible configurations
Barometer BPM085 XOR	MS5611	2
Camera RGB XOR	Infrared	2
Servos Aileron OR	Rudder	3
Total (Combined)	-	12

Table 2: Coverage criteria compared in the test case generation

Coverage	Number of generated products	Percentage of SPL products covered (%)	Errors found
Total	12	100	11
Partitioned	9	75	8
Partitioned	6	50	5
Partitioned	3	25	2

Table 3: Results of the test of Step 1 of the transformation, after the execution of the test cases from the Total coverage criteria

Test case	Result	Error found
Variant0	Error	org.eclipse.emf.ecore.xmi.UnresolvedReferenceException
Variant1	Error	org.eclipse.emf.ecore.xmi.UnresolvedReferenceException
Variant2	Pass	-
Variant3	Error	org.eclipse.emf.ecore.xmi.UnresolvedReferenceException
Variant4	Error	org.eclipse.emf.ecore.xmi.UnresolvedReferenceException
Variant5	Error	org.eclipse.emf.ecore.xmi.UnresolvedReferenceException
Variant6	Error	org.eclipse.emf.ecore.xmi.UnresolvedReferenceException
Variant7	Error	org.eclipse.emf.ecore.xmi.UnresolvedReferenceException
Variant8	Error	org.eclipse.emf.ecore.xmi.UnresolvedReferenceException
Variant9	Error	org.eclipse.emf.ecore.xmi.UnresolvedReferenceException
Variant10	Error	org.eclipse.emf.ecore.xmi.UnresolvedReferenceException
Variant11	Error	org.eclipse.emf.ecore.xmi.UnresolvedReferenceException

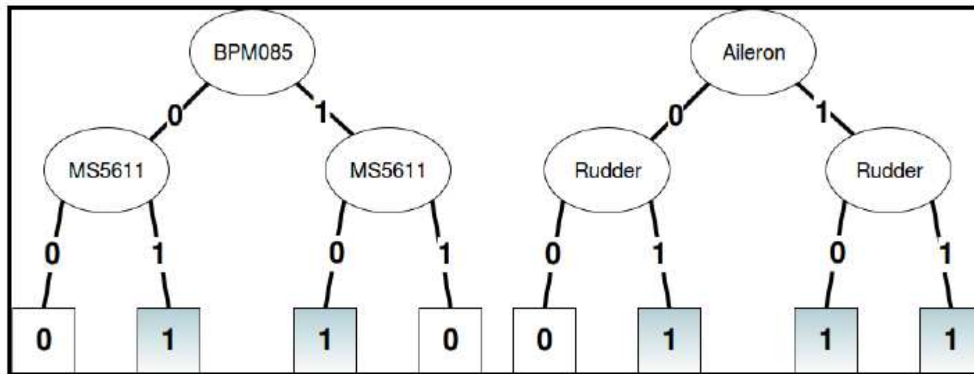

Fig. 8: Analysis of "Barometer" and "Servos" variability

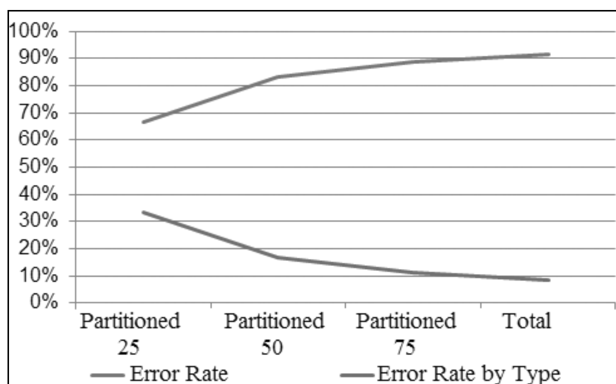

Fig. 9: Results in test execution considering step 1 of the transformation of SyMPLES

Several test cases showed the presence of errors, in the tests of Step 1 of the transformation. Figure 9 presents a comparison between the coverage criteria and provides an analysis of the type of error found. The error rate is calculated based on the amount of errors found divided by the amount of test cases. The error rate by type is calculated based on the quantity of error types divided by the amount of test cases.

Increasing the number of test cases resulted in a gain in the number of errors found (upper curve in the graph, Fig. 9). In other words, the blue curve represents the variation of the number of test cases that showed an error between the coverage criteria. The coverage criteria applied, showed in Fig. 9, was the types classified as partitioned or total. For an example, in the "Partitioned 25" almost 70% of the test cases showed an error.

Although the results showed a high error rate, only one type of error was detected. The type of error identified was the Type 5 of the classification presented in the Fourth section. This is showed by the bottom curve in the graph. Increasing the number of test cases did not result in the detection of other types of errors (Fourth section, Validation of Model Transformations). Therefore, in this case, the "Partitioned 25" criterion is more efficient because its error rate by different type is higher than others, about 33%.

All the errors identified are related to unresolved references of elements in the input models (UnresolvedReferenceException). Therefore, they were classified as type 5: Common coding errors. Only after the correction of the errors, the test was executed again and no error was found in the transformation. Without errors, the intermediary models were successfully generated.

With the intermediary models generated by the transformation, the test of the second step of the transformation (Step 2: Generate Functional Blocks) was performed, but no error was detected, which means that all of the output models were generated.

Then, the validation proceeds with the dynamic test. Each output model was executed with the dynamic test script and no error was identified, allowing the visualization of the Simulink models, ending the Run test activity.

Comparing Results

Summarizing the results from this case study, the proportion of errors found by test case generated obtained was about 92%, considering total coverage of the SPL. This means the effectiveness of the test case set: 11 from 12 test cases were useful to find errors. However, these errors are related to common coding problems. No error was found in the dynamic test. In addition, errors of Type 1-4 (Fourth section), which are specific of the MDE context, could not be found.

Due to the specific nature of this case study, which involves the test of model transformation of the SyMPLES approach and test case generation from an SPL specified with SyMPLES, it is hard to compare results to evaluate the generation technique.

However, despite the differences in the techniques, some concepts are similar when generating test cases. Sen *et al*. (2009), for an example, presented a test case generation algorithm based on the metamodel. Using a metamodel-based technique, the test case set generated is more generic, normally bigger and they also applied coverage criteria to minimize the test size, but including mutation factors and other strategies. The mutation factors refer to how much distinguish are the test cases in the test case set. A higher mutation factor could increase the probability of the test case find an error (Sen *et al*., 2009).

On the other hand, using an SPL-based technique the generation is specific to the possible configurations of the products. Therefore, the mutation between the test cases is limited to the quantity of the variabilities in the SPL.

A simple approach to analyze the mutation score between the test cases, in this context, can be done as follows: divide the sum of the variations by the sum of the core elements of the SPL. Considering the SPL specification, composed of a set of diagrams (including the diagram presented in Fig. 2), the sum of core elements in this case study is 7 blocks (in the block definition diagram) plus 21 blocks (in internal diagrams) (Fragal *et al*., 2013).

From such 28 blocks obtained from the specification, the variabilities are located on only 6 blocks and thus the mutation score in this context would be about 21%. In addition, as the XOR variabilities require one block only in each product, the changes fall, in practice, to 4 blocks with ~14% mutation score. Comparing to the metamodel-based technique (Sen *et al*., 2009), the mutation score achieved is up to 87%. This can be explained due to the generic and big test case set in this technique.

Previous Work

Before presenting the comparison with the previous work, it is worthy to mention that the same version of the SyMPLES transformation was used. Table 4 shows a summary of the results compared with previous work results. The coverage criterion compared is the total coverage, in both techniques. The errors of Type 5 were found applying the test case generation based on the SPL and errors of Type 3 were found using generation based on the SysML metamodel.

In order to compare the effectiveness of the test case sets in both techniques, the graph in Fig. 10 is presented. The graph leads to the conclusion that the generation based on SPL was more efficient in the validation of the transformation. However, only errors of Type 5 were found, which are classified as common coding errors.

On the other hand, the generation based on metamodel was capable to find errors of Type 3, which are more related to the development of the MDE transformations. In this case study, the test case set effectiveness is higher with the SPL-based technique, but the quantity of errors found is higher with the metamodel based technique.

It is worthy to highlight that errors of Type 4 (Ambiguity) are difficult to find using black-box testing approach and errors of Type 2 require information from the output metamodel, although they can be detected using dynamic testing when applicable (Tiso *et al*., 2012).

Threats to Validity

A threat to validity identified is that the SPL used was relatively small (only 12 potential products) in this case study and domain specific, related to the Embedded Systems context. Using total coverage on the SPL was possible, but this coverage could be difficult to apply. For instance, an SPL with *n* features can yield up to *2n* individual systems (or products) (Classen *et al*., 2011). Examples of bigger SPLs include: Linux 2.6.32.2 kernel, with 6052 features (Peng *et al*., 2013) and Eclipse SPL, 1024 features (Johansen *et al*., 2012).

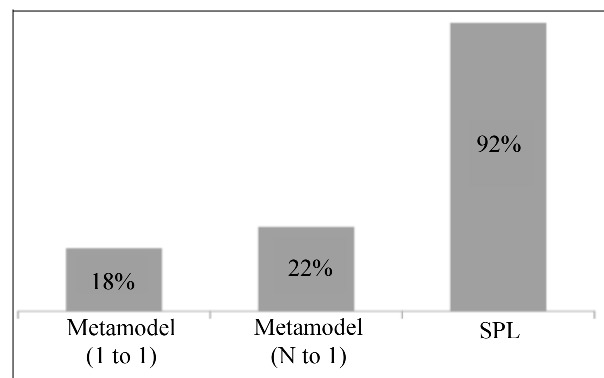

Fig. 10: Comparison between SPL-based and metamodel-based techniques, regarding the effectiveness of the test case set generated

Table 4: Comparison of results with the previous work (Giron *et al*., 2017)

Generation technique	Generation policy	Maximum amount of test cases	Type of error found	Errors found
SPL-based	-	12	Type 5	11
Metamodel based	1 to 1	184	Type 3	33
	N to 1	46	Type 3	10

Conclusions, Contributions and Future Work

The validation of MDE transformations is required for quality assurance. In this study, a case study for validation based on the functional test of a model transformation was presented. An SPL-based technique was used for test case generation and made it possible to identify errors in the transformation, in a systematic way, contributing to the quality of the transformation.

The main contribution of this case study is to provide an evaluation of the technique of the SPL-based test case generation, in the context of MDE transformations. A motivation to the case study was the fact the test cases generated would be more related to the transformation of the SyMPLES approach, which uses SPL concepts. The hypothesis suggested that test would produce better results, compared to other test case generation techniques, like the metamodel-based. However, with specific test cases generated, it was only identified one type of error (Type 5). Although this technique obtained a higher rate of errors identified in the transformation, supporting the hypothesis proposed, more evidence would be necessary to compare and evaluate SPL-based and Metamodel-based techniques.

Another contribution is the validation of the SyMPLES model transformation. The provided information and the tools developed helped to improve the model transformation.

The tools developed in this study aimed to automate the validation activities; however, they are specific to the transformation under test. For example, the test case generation tool can be applied to SPL specified with SyMPLES stereotypes. In general, the concepts are generic but the tools developed are specific to the transformation under test. The high variety of technologies related to MDE transformations can make it difficult to reuse such tools.

Directions for future work would include the investigation of methods of structural testing in the validation of the transformation of the SyMPLES approach. Combining with structural testing the validation level could be higher.

Funding Information

The authors thank CAPES foundation for partially funding this work, to Vanderson Fragal for his support and contributions and to CNPq for the support to the INCT-SEC project.

Author's Contributions

All authors contribute equally to this work.

Ethics

This paper provides an original contribution of the authors and it is not published elsewhere. All refereces used are cited in this paper. There is no ethical issue involved in this article.

References

Anand, S., E.K. Burke, T.Y. Chen, J. Clark and M.B. Cohen *et al.*, 2013. An orchestrated survey of methodologies for automated software test case generation. J. Syst. Software, 86: 1978-2001. DOI: 10.1016/j.jss.2013.02.061

Benavides, D., S. Rueda, P. Trinidad and A. Cortés, 2007. FAMA: Tooling a framework for the automated analysis of feature models. Proceedings of the 1st International Workshop on Variability Modelling of Software-Intensive Systems, (SIS' 07), Limerick Irlanda, pp: 129-134.

Beuche, D., 2012. Modeling and building software product lines with pure::variants. Proceedings of the 16th International Software Product Line Conference, Sept. 02-07, ACM, Salvador, Brazil, pp: 255-255. DOI: 10.1145/2364412.2364457

Biehl, M., Sjöstedt, C. J. and Törngren, M., 2010. A modular tool integration approach: Experiences from two case studies. Proceedings of the 3rd Workshop on Model-Driven Tool and Process Integration, (TPI' 10).

Brottier, E., F. Fleurey, J. Steel, B. Baudry and Y. Le Traon, 2006. Metamodel-based test generation for model transformations: an algorithm and a tool. Proceedings of the 17th International Symposium on Software Reliability Engineering, Nov. 7-10, IEEE Xplore Press, Raleigh, NC, USA, pp: 85-94. DOI: 10.1109/ISSRE.2006.27

Classen, A., P. Heymans, P.Y. Schobbens and A. Legay, 2011. Symbolic model checking of software product lines. Proceedings of the 33rd International Conference on Software Engineering, May 21-28, ACM, Waikiki, Honolulu, HI, USA, pp: 321-330. DOI: 10.1145/1985793.1985838

Fawaz, K., F. Zaraket, W. Masri and H. Harkous, 2015. PBCOV: A property-based coverage criterion. Software Quality J., 23: 171-202. DOI: 10.1007/s11219-014-9237-3

Fleurey, F., J. Steel and B. Baudry, 2004. Validation in model-driven engineering: Testing model transformations. Proceedings of the 1st international Workshop on Model, Design and Validation, Nov. 2-2, Rennes, France, IEEE Xplore Press, pp: 29-40. DOI: 10.1109/MODEVA.2004.1425846

Fragal, V.H., R.F. Silva, I.M.S. Gimenes and E. OliveiraJr, 2013. Application engineering for embedded systems transforming SysML specification to simulink within a product line based approach. Proceedings of the 15th International Conference on Enterprise Information Systems (EIS' 13), pp: 94-101.

Friedenthal, S., A. Moore and R. Steiner, 2009. A Practical Guide to SysML: The Systems Modeling Language. 1st Edn., Morgan Kaufmann, Boston, ISBN-10: 012378607X, pp: 560.

Giron, A.A., I.M.S. Gimenes and E. OliveiraJr, 2017. Case study of test case generation based on metamodel for model transformations. J. Software, 12: 364-378. DOI: 10.17706/jsw.12.5.364-378

Guerra, E., 2012. Specification-driven test generation for model transformations. Proceedings of the 5th International Conference on Theory and Practice of Model Transformations, May 28-29, Springer, Prague, Czech Republic, pp: 40-55. DOI: 10.1007/978-3-642-30476-7_3

Johansen, M.F., Ø. Haugen and F. Fleurey, 2012. Bow tie testing: A testing pattern for product lines. Proceedings of the 16th European Conference on Pattern Languages of Programs, Jul. 13-17, ACM, Irsee, Germany. DOI: 10.1145/2396716.2396725

Jouault, F. and I. Kurtev, 2005. Transforming models with ATL. Proceedings of the International Conference on Model Driven Engineering Languages and Systems, Oct. 02-07, Springer, Berlin, Heidelberg, pp: 128-138. DOI: 10.1007/11663430_14

Küster, J. M. and Abd-El-Razik, M., 2006. Validation of model transformations: First experiences using a white box approach. Proceedings of the International Conference on Model Driven Engineering Languages and Systems, Oct. 01-06, Springer, Berlin, pp: 193-204. DOI: 10.1007/978-3-540-69489-2_24

Lano, K., T. Clark and S. Kolahdouz-Rahimi, 2015. A framework for model transformation verification. Formal Aspects Comput., 27: 193-235. DOI: 10.1007/s00165-014-0313-z

Lin, Y., J. Zhang and J. Gray, 2005. A Testing Framework for Model Transformations. In: Model-Driven Software Development, Beydeda, S., M. Book and V. Gruhn (Eds.), Springer, Berlin, Heidelberg, pp: 219-236.

Lochau, M., S. Oster, U. Goltz and A. Schürr, 2012. Model-based pairwise testing for feature interaction coverage in software product line engineering. Software Quality J., 20: 567-604. DOI: 10.1007/s11219-011-9165-4

Marvedel, P., 2003. Embedded System Design. 1st Edn., Springer, Berlin, ISBN-10: 1402076908, pp: 241.

Mathworks, 2017. MATLAB Simulink.

Mellor, S.J., 2004. MDA Distilled: Principles of Model-Driven Architecture. 1st Edn., Addison-Wesley Professional, Boston, ISBN-10: 0201788918, pp: 150.

Mendonca, M., M. Branco and D. Cowan, 2009. SPLOT: Software Product Lines Online Tools. Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications, Oct. 25-29, ACM, Orlando, Florida, USA, pp: 761-762. DOI: 10.1145/1639950.1640002

OliveiraJr, E., I.M.S. Gimenes and J.C. Maldonado, 2010. Systematic management of variability in UML-based software product lines. J. Univ. Comput. Sci., 16: 2374-2393. DOI: 10.3217/jucs-016-17-2374

Peng, X., Z. Xing, X. Tan, Y. Yu and W. Zhao, 2013. Improving feature location using structural similarity and iterative graph mapping. J. Syst. Software, 86: 664-676. DOI: 10.1016/j.jss.2012.10.270

Sen, S., B. Baudry and J.M. Mottu, 2009. Automatic model generation strategies for model transformation testing. Proceedings of the International Conference on Theory and Practice of Model Transformations, (PMT' 09), Springer, Berlin, Heidelberg, pp: 148-164. DOI: 10.1007/978-3-642-02408-5_11

Silva, R., V. Fragal, E. OliveiraJr, I.M.S. Gimenes and F. Oquendo, 2013. SyMPLES: A SysML-based approach for developing embedded systems software product lines. Proceedings of the 15th International Conference on Enterprise Information Systems, (EIS' 13), Angers, France, pp: 257-264.

Tiso, A., G. Reggio and M. Leotta, 2012. Early experiences on model transformation testing. Proceedings of the 1st Workshop on the Analysis of Model Transformations, Oct. 02-02, ACM, Innsbruck, Austria, pp: 15-20. DOI: 10.1145/2432497.2432501

Van der Linden, F., K. Schmid and E. Rommes, 2007. The Product Line Engineering Approach. In: Software Product Lines in Action, van der Linden, F.J., K. Schmid and E. Rommes (Eds.), Springer Berlin Heidelberg, ISBN-10: 3540714375, pp: 3-20.