2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C) | 978-1-6654-7836-6/21/$31.00 ©2021 IEEE | DOI: 10.1109/QRS-C55045.2021.00162

Heterogeneous Modeling and Testing of Software Product Lines

Fevzi Belli*
University of Paderborn
Paderborn, Germany
belli@upb.de

Tugkan Tuglular
Izmir Institute of Technology
Izmir, Turkey
tugkantuglular@iyte.edu.tr

Ekincan Ufuktepe
University of Missouri - Columbia
Columbia, MO, USA
euh46@missouri.edu

***Abstract*—Software product line (SPL) engineering is a widely accepted approach to systematically realizing software reuse in an industrial environment. Feature models, a centerpiece of most SPL engineering techniques, are appropriate to model the variability and the structure of SPLs, but not their behavior. This paper uses the idea to link feature modeling to model-based behavior modeling and to determine the test direction (top-down or bottom-up) based on the variability binding. This heterogeneous modeling enables a holistic system testing for validating both desirable (positive) and undesirable (negative) properties of the SPL and variants. The proposed approach is validated by a non-trivial example and evaluated by comparison.**

***Index Terms*—software product line engineering, model-based testing, holistic testing**

I. Introduction

A software product line (SPL) is a collection of software products developed from a single architecture [1], [2]. A SPL is used to produce a product family, i.e., products with similar features. The objective of software product line engineering (SPLE) is to systematically develop SPLs to enable the reusing of components for production at lower costs, faster, and increased quality than the developing them from scratch [3], [4].

SPLE combines two development processes. *Domain engineering* focuses on the identification of variable features shared by the family of products from the SPL represented by a feature model (FM). The counterpart of domain engineering, *application engineering*, aims the generation of products from the SPL with respect to the product configuration, where all the variabilities are bound. Application engineering maps features identified by the product configuration onto the final product.

When a product is generated from an SPL, how would someone confirms that it contains all the features and the features alone and together work as expected. To this end, one approach would be model-based SPL testing. In such a case, systematic, model-based SPL testing is required to confirm that every product will meet its requirements for the features it is composed of and will behave as its potential user anticipates. To achieve this objective, a heterogeneous modeling approach using relevant modeling techniques is necessary. Therefore, the chosen heterogeneous modeling approach should be able to cover both *feature model* and *behavior model*.

*Authors are ordered alphabetically

FM is the de facto standard for modeling the variability and structural aspects of SPL. The behavioral aspect can be handled, however, by models of many categories, for example, by a variety of UML diagrams, but also formal models such as finite state machines and Petri nets.

Weißleder and Lackner suggested linking FM to behavior modeling to control the variability binding while deriving product variants of the SPL [5]. This view enables also systematizing the test process focusing on the architecture of the SPL or the variant to be derived. The former DE-centric view leads to a bottom-up test strategy while the latter leads to a product- and AE-centric top-down test strategy. The present paper refines and extends this heterogeneous modeling approach of Weißleder and Lackner.

The novelty of this paper stems from that it enables systematic testing of the SPL and variants in a holistic way: *Positive testing* to validate that the software under consideration does everything it is supposed to do and *negative testing* to validate that it does not do anything it is not supposed to. The idea behind holistic testing is to complement the given model to enable negative testing.

Weißleder and Lackner use Statecharts to model the behavior of the SPL and variants. They suggest transition coverage of Statechart for positive testing. The present paper suggests event sequence graphs (ESGs) for modeling that can easily be complemented to enable negative testing. The concept of coverage of events and event sequences of increasing length will be applied to positive and negative testing in order to scale the test process.

The variability aspect will be discussed in the Section II, along with a summary of the terms and notions used in the paper, especially SPL modeling at different levels (150% and 100% models) and event sequence graphs for behavior modeling. The concept of linking both, variability modeling and behavioral modeling, will be explained and illustrated on a running example in Section II. Binding of the variability and domain-centric and product-centric testing will be discussed and illustrated on a running example in Section III. The concept of event and event sequence coverage of the holistic testing view will be discussed in Section III as well. A non-trivial example will be used in Section IV for evaluation of the approach, analyzing its characteristics, and its validation. Moreover, the present approach will be compared with a

2693-9371/21/$31.00 ©2021 IEEE
DOI 10.1109/QRS-C55045.2021.00162

similar approach based on the case study. A self-critical discussion, covering also threats to the validity, concludes Section IV. Section V continues and extends the discussion of the related work started in Section II. The results of the paper and planned follow-on research are briefly discussed in Section VI.

II. Background

For model-based testing, a SPL should have its structure and behavior modeled and these two models should be molded. For structure modeling, a feature model is used. A feature model not only expresses features but also feature variability, i.e., variation points, within a product family. Grönniger et al. call this model *150%* [5]–[8]. A feature model becomes a product configuration if it models a product variant of the product family. Grönniger et al. call this model *100%* [5]–[8]. Figure 1 shows the 150% feature model, whereas Figure 2 describes the structure of a product, or variant, derived from the 150% feature model given in Figure 1.

A behavioral model represents interactions with users and other features. Two possible behavioral models are statecharts and event sequence graphs (ESGs). In this paper, both are used to model behavior of features and product variants. Like feature model, a behavioral model is called *150%* model if it represents the whole SPL and *100%* model if it represents a product [6].

An ESG is, very briefly and informally explained, a directed graph, of which nodes semantically represent events connected by arcs that realize "follow" relation, whereby a double arrow a loop between two nodes represents (see the example in Figure 3). An ESG is actually a simplified form of a finite state machine, where states and input/outputs are merged and can be viewed as semantically enriched Myhill Graphs [9].

The definitions of even sequence graphs given below are taken from [10] and [11].

Definition 1. An *event sequence graph (ESG)* is a directed graph where $V = \emptyset$ is a finite set of nodes (vertices) and $E \subseteq VxV$ is a finite set of *arcs (edges)* and $\Xi, \Gamma \subseteq V$ finite sets of distinguished vertices with $\xi \in \Xi, \gamma \in \Gamma$ called entry nodes and exit nodes, respectively [10].

Definition 2. Let (V, E) be an *ESG*. Then a sequence of vertices $< v_0, ..., v_k >$ is called an *event sequence (ES)* if the sequence is a walk on *ESG* [10].

Each edge of an ESG represents a legal event pair, or simply, an *event pair (EP)*. ES $< v_i, v_k >$ of length 2 is an EP [10].

Definition 3. An ES $< v_0, ..., v_k >$ is called a *complete event sequence (CES)*, if $v_0 = \xi \in \Xi$ is the entry and $v_k = \gamma$ is the exit. A CES represents a *test sequence* [11].

Definition 4. A faulty ES is a *complete* (or, it is called a *complete faulty event sequence, CFES*) it $\alpha(faulty\ ES) = \xi \in \Xi$ is an entry. The ES as part of a CFES is called *starter* [11].

Note that ***Definition 4*** explicitly points out that a CFES does not finish at an exit, unlike an CES that must finish at an exit [11].

This lean modeling makes learning and using ESGs simple. It prevents the designers make modelling errors. Moreover, since an ESG is a directed graph, some algorithms of graph theory can be used for test case generation and test optimization ([11]–[14] for more details). In addition, ESG modeling enables to perform the negative testing easily and efficiently by completing the original ESG by adding missing edges that represent *illegal* user-system interactions, likely leading to unexpected or undefined system reactions. Negative testing using ESG will be explained in Section III-A in more detail.

Figure 3 depicts the "online shop" behavior model corresponding to the 150% model in Figure 1 as an event sequence graph. The model in Figure 3 attempts to present all possible behavioral alternatives of the SPL, represented as sequences of events. Thus, it is a 150% model. Specific variants, as 100% models, are to be derived from this model. Note, however, this running example, strongly simplifies the reality, ignoring many likely events, for example, the case the card is empty after removing all the products, or the case to enter in an infinite loop if there an invalid payment. On the other hand, it is possible, even likely, that no variant makes sense or is feasible to be implemented to realize the 150% model. Figure 4 illustrates the coupling of a feature model (Figure 3) and an ESG. Here, where the feature "Credit Card" of the feature model refers to the node "Select Credit Card" of the ESG to signal the necessity of the high security.

III. Heterogeneous SPL Testing

An SPL can be tested either before or after the binding of the variability. The next two subsections discuss these two strategies using Lackner et al.'s approach [5], [15]–[17] (See Figure 5). *Domain-centric testing* requires the binding of the variability *after* generating a test suite for the SPL, while *product-centric testing* requires the binding of the variability *before* generating a test suite for each product [3]. Figure 5 outlines both strategies; each one consists of two subsequent steps [3]:

- Domain-Centric Testing: PLCT1 (product line testing1) is used to generate SPL test cases from the SPL test model, whereas PLCT2 (product line testing2) is used for variability binding and test generation for product variants.
- Product-Centric Testing: PCT1 is used for the variability binding and product test model generation, and PCT2 is used for the generation of the product test cases from the product test model.

A. Domain-Centric Bottom-Up Testing

Domain-centric testing of an SPL links the SPL's 150% feature model and 150% behavioral model to produce all possible test cases [3]. We relate the behavioral model in Figure 3 to the feature model in Figure 1 using the concept presented in Section II. This linkage will guide test generation for the product under consideration.

The "search" feature is optional, meaning it can be included or excluded. As a result, we'll require a "guard" that may

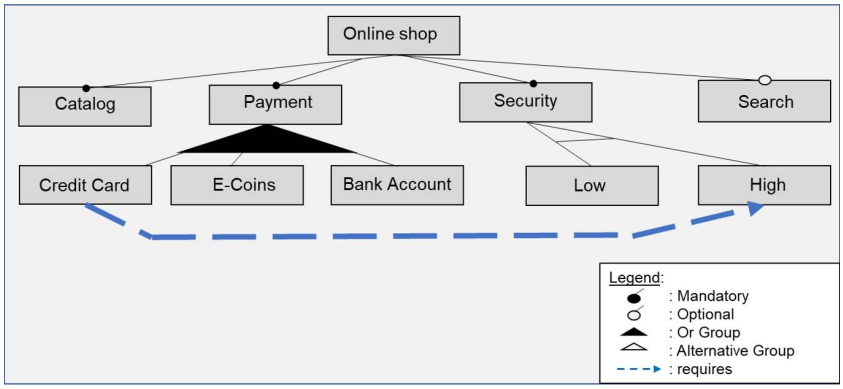

Fig. 1: Example Feature Model as a 150% Model of Online Shop SPL [5]

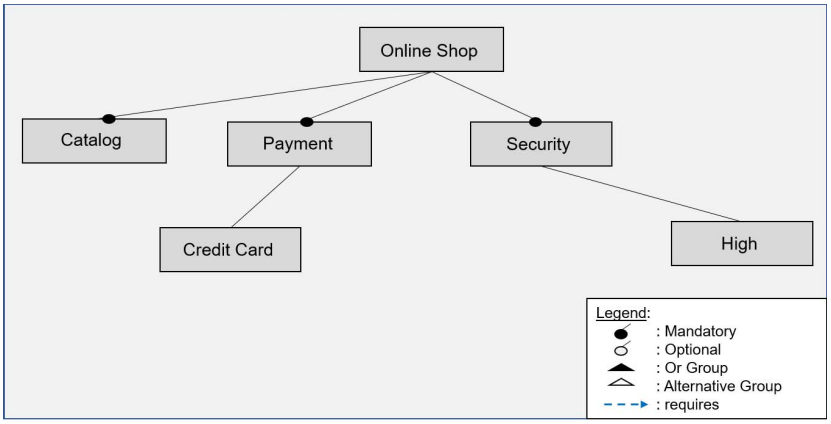

Fig. 2: Deriving a Variant as a 100% Model out of the 150% Model of Online Shop SPL [3]

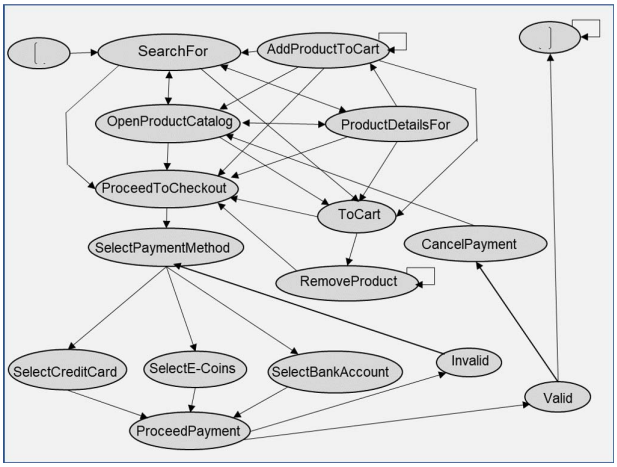

Fig. 3: Behavior Model as a 150% Model for the Example in Figure 1 [3]

be selected when deriving variants with this feature. This guard will be deselected for variations that do not include the search feature. The guard mechanism will be used to handle variability in product development.

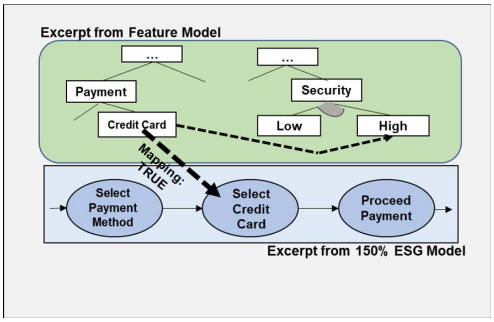

Fig. 4: Linking FM with ESG [3]

The example ESG in Figure 3 can now be extended by adding *faulty event pairs* (*FEP*) (dotted lines), that are illegal sequences of events, leading to Figure 7 (The FEPs represent user-system interactions that violate the specification, thus modeling faults). *Negative test cases* start also at "[", and contain *faulty event pairs* (*FEP*), forming a complete faulty ES (CFES). An CFES is supposed to never reach the exit, symbolized by "]", as it contains a faulty sequence of events. Further, a CFES (as an already faulty sequence of events)

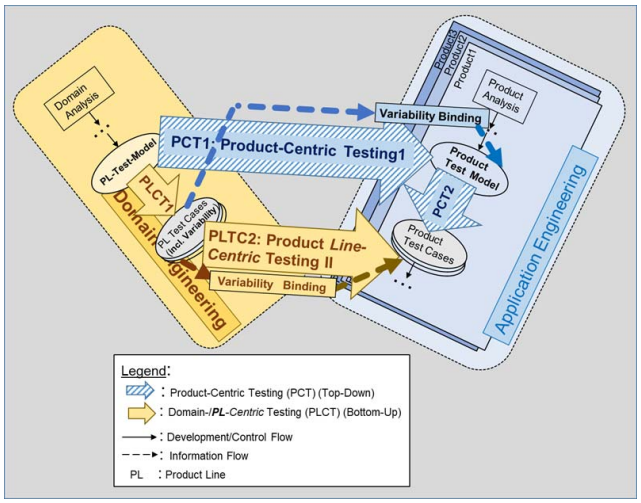

Fig. 5: Domain-Centric vs. Product-Centric Testing [3]

cannot contain any faulty FES of the length more than 2 as a fault cannot be "faultier". Thus, we consider CFES having only a single fault.

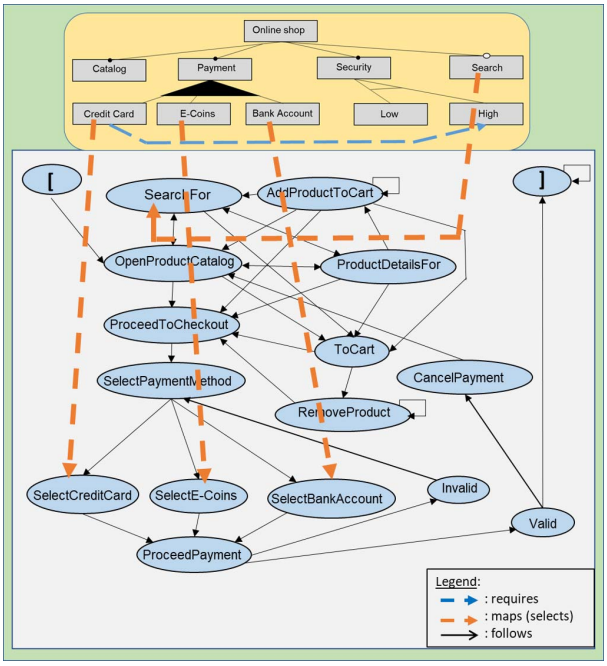

Fig. 6: Domain-Centric Testing of the Running Example, 150% ESG Model [3]

Test Suite Designer (TSD) [18] is utilized to generate test sequences for the ESG given in Figure 3 (relating to Statecharts in Figure 6 and Figure 7) and they can be accessed at (https://github.com/esg4aspl/SPL-ESG-Examples/tree/main/OnlineShop) and consist of a test suite to cover all EPs for positive testing and all *FEP*s for negative testing.

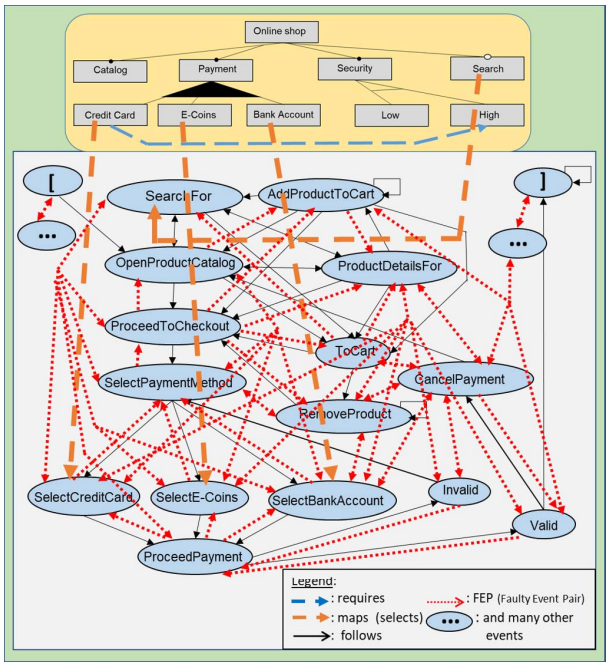

Fig. 7: dto. as Completed ESG Model, extended by Faulty Event Pairs (FEP) [3]

B. Product and Application-Centric Top-Down Testing

Product-centric testing of an SPL begins with the 150% structural and behavioral models (Figure 3) and links it with the 100% feature model (Figure 4). 100% models can be generated for the running example using the ESG in Figure 3 and using the Statechart of the variant in Figure 4 by deselecting some features (as discussed in sections II and III-A; [19]).

An example of using the mapping function in Figure 4 for product-centric testing would be deselecting search and low security features, which leaves only credit card payment feature. Figure 8 and 9 represent both variants as sub-graphs of the 150% model. Thus, covering both of the variants is necessary to reach full coverage of events, event pairs, etc.

Test suites generated by the TSD for the ESG given in Figure 8 and 9 (as augmented with FM in Figure 7) can be accessed at (https://github.com/esg4aspl/SPL-ESG-Examples/tree/main/OnlineShop).

C. Advantages of Holistic Testing

The symbiosis of positive and negative testing has many advantages that will be demonstrated by following examples (Figure 7).

- A positive test case, for example, *pt1* given by the CES *[OpenProductCatalog SearchFor ProductDetailsFor ToCart ProceedToCheckOut SelectPaymentMethod SelectCreditCard ProceedPayment Valid]* checks a regular case of buying a product variant and payment by credit card. Any variant has to accomplish this task as a requirement.

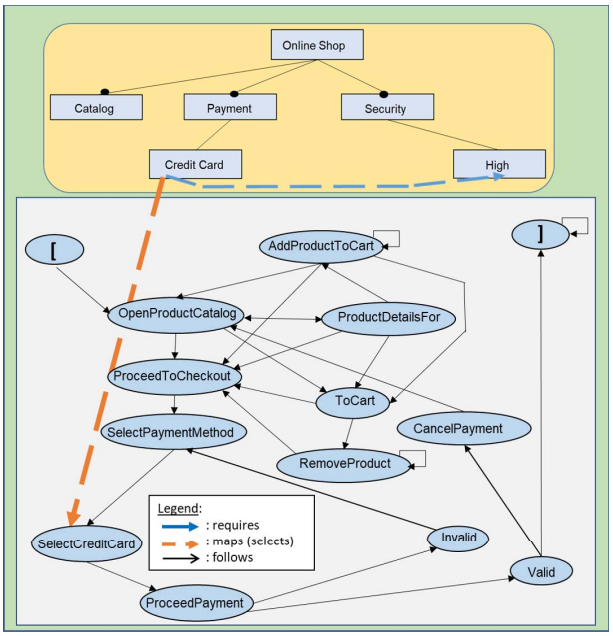

Fig. 8: Product-Centric Testing of the Running Example, Variant 1 - Credit Card Payment Only [3]

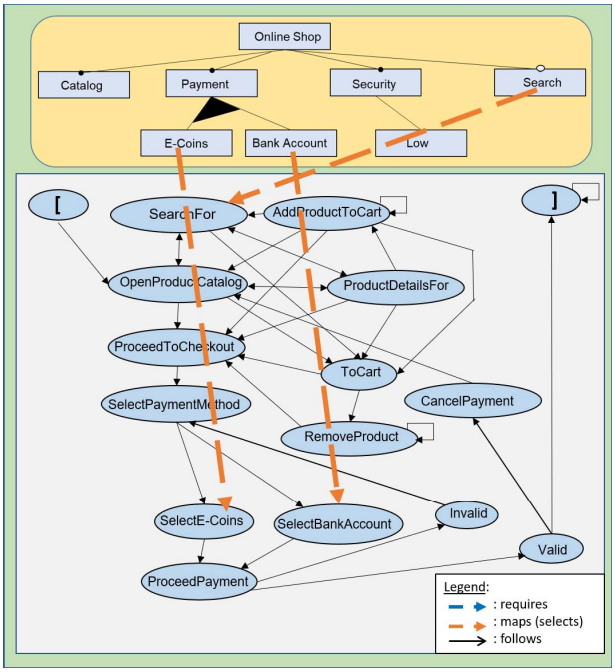

Fig. 9: Product-Centric Testing of the Running Example, Variant 2 – E-Coin and Bank Account Payment [3]

- A negative test case, for example, *nt1* given by the FCES *[OpenProductCatalog SearchFor* ***ProductDetailsFor CancelPayment*** forms an irregular, illegal case of canceling a payment without having proceeded to check out, and then selecting a payment method (FEP this CFES includes is underscored).
- Another negative test case *nt2* given by the FCES *[OpenProductCatalog SearchFor ProductDetailsFor ToCart* ***ProceedToCheckOut SearchFor*** checks an interesting case of attempting to continue buying after proceeding to check out (Again, its FEP is underscored). The present model excludes this case, which might be, however, not in the sense of the business that might rather encourage to sell more products. Thus, Figure 7, as a specification, needs to be corrected to comply with the business policy, making *ProceedToCheckOut SearchFor* a legal EP.

To sum up, the holistic testing can be used not only to validate the implementation, but also to validate the requirements, long before the implementation starts.

D. Test Suite Comparison

Weißleder and Lackner reported that using the Bottom-up approach they manually created a single test case, 27 steps in total [5]. However, with the Conformiq Designer [20] they generated twelve test cases, with 59 event calls. The TSD for ESGs generates six CESs with 60 events. In Table I, we have reproduced the test cases from Weißleder et al. [5] with Conformiq Designer and presented results with our ESG approach results for Online Shop. Although results are close for the event coverage, for event-pair coverage ESG approach with the TSD tool outperforms Statechart approach with the Conformiq Designer, because ESG approach optimizes the test sequences with minimum number of events whereas the Statechart approach prefers more test sequences with shorter sequences. For detailed explanation, please refer to [3].

IV. Validation and Discussion

In this section, we first introduce our Smart Home SPL case study. Then, we share the results of the generated test cases for both Statechart and ESG approaches using Bottom-Up Testing. We used the Bottom-Up testing since Weißleder et al. [5] and our running example results in Table I have shown the number of generated test cases and events is less than the Top-down Testing.

To generate the test cases from a Statechart diagram. We used for the Statecharts the semantics model as introduced in [21] that is similar to the UML Statechart 5.0.0 version of Conformiq Designer [20] for generating test cases from Statechart, using the *All Transitions* and *Transition Pairs* coverage criterion. For event coverage, we have simply developed an algorithm that finds the shortest path between the start event ("[" node) to the target event and concatenates with the path between the target and end event ("]" node). This process is continued until all events are covered. The main goal of the event coverage strategy is to generate smaller path sizes of test cases. Finally, we perform a simple test suite minimization to over the generated test cases. The test suite minimization strategy checks every test case and removes the test case if its covering events are covered by the entire other cases. We have made source codes publicly available for

TABLE I: Online Shop test case and number of events

Test Type		Positive/Negative Test	Coverage Type	Statechart Approach		ESG Approach	
				#test cases	*#events*	*#test cases*	*#events*
Bottom-Up Testing		Positive Test	Trans./Event Cov.	12	59	6	60
			Trans./Event Pair Cov.	42	184	1	61
		Negative Test	-	n/a	n/a	205	858
Top-down Testing	Var. 1	Positive Test	Trans./Event Cov.	12	59	6	60
			Trans. Pair/Event Pair Cov.	42	184	1	61
		Negative Test	-	n/a	n/a	205	858
	Var. 2	Positive Test	Trans./Event Cov.	11	42	6	60
			Trans. Pair/Event Pair Cov.	40	172	1	61
		Negative Test	-	n/a	n/a	177	731

event coverage and event pair coverage which can be found at (https://github.com/esg4aspl/esg-engine).

In addition, we compare the Statechart and ESG approaches based on the number of generated test cases, events, and negative testing capability. Finally, we discuss the potential threats to validity of our study.

A. Smart Home SPL

In this section, the proposed approach is evaluated on the smart home SPL obtained from software product lines online tools repository [22]. The chosen smart home SPL is modified to possess the following features:

- mandatory features: User Interface (UI), Windows Management, Light Management
- optional features: Blinds Management, Alarm, Security, Fire Control, Presence Simulation

The user interface feature has a mandatory touchscreen and optional web, or mobile user interface accessed over the Internet. All windows management, light management, and blinds management features have a mandatory manual operation as well as optional automatic operation. The Presence simulation feature has optional light, blinds, and audio-video operations executed in presence of a person. The Alarm feature provides light, siren, and bell types of alarms manually and automatically. The Alarm feature is utilized by security and fire control features. Security is implemented through an authentication device, which can be a keypad, retina scanner, or fingerprint reader, and optionally through an intrusion detection device. Fire control feature provides calling the fire department, running bell alarm as well as operating fire sprinklers.

The feature model of smart home SPL is generated using Feature IDE [23] and shown in Figure 10. The minimum mandatory product configuration for smart homes is composed of a touchscreen user interface, manual window management, and manual light management and can be considered as the base product. 150% smart home product configuration contains all features. The Statechart of the 150% product configuration is given in Figure 10, whereas the ESG of the 150% product configuration is shown in Figure 11. Both Statechart and ESG models are prepared in a way that enables modelers to easily remove or add features using, for instance, guards.

The TSD tool that inputs smart home ESG generates negative test cases along with positive test cases. Ten interesting examples of negative test cases for the smart home software product under consideration are selected among 2767 test cases generated by TSD tool and shown in Table II.

TABLE II: Examples of negative test cases for the smart home product under consideration

Test Number	Test Case
1	right after blinds are opened automatically, fire sprinklers are turned on automatically
2	right after motion detector activated, fire detected
3	right after presence detected, inhouse light is turned off automatically,
4	right after daylight is detected, fire sprinklers are turned on automatically
5	right after smoke detector activated, fire department is called,
6	right after a fire is detected, windows are opened automatically
7	right after glass break sensor is activated, windows are opened automatically
8	right after house intrusion detected, blinkers are turned on automatically,
9	right after authentication is successful, the siren is turned on automatically
10	right after presence at home detected, fire sprinklers are turned on automatically

B. Results and Discussion

We present the generated test cases from the Statechart and ESG approaches. Table III shows the number of generated test cases and the number of total events of the test suite for both approaches. The first column represents the testing type. The second column represents the performed positive and negative test independent from each other for the selected testing type. The third column represents the coverage type for the corresponding testing type. There are two coverage types *Transition/Event Coverage* and *Transition/Even Pair Coverage*. Since that we mentioned Statecharts can be projected to ESG, the *Transitions* in the Statechart corresponds to the *Events*, both are shown in a single row. This also applies for *Transition Pairs* and *Event Pairs* as well. The forth column represents the generated test case information for the *Statechart Approach*, which includes the number generated test cases, and the total number *events/transitions* in the total test suite. The fifth column represents the generated test case information for the *ESG Approach*, which also includes the total generated test cases and the number of *events/transitions* in the test suite. The results are compared within three criteria:

number of test cases, number of events, and generated negative tests.

In terms of generated test cases based on 100% transition coverage, or in other words 100% event coverage, the Statechart approach has generated 42 test cases, and ESG has generated fewer tests with 33 test cases. However, when the cost of events of the test suites is compared, the ESG event coverage is expensive than the Statechart approach. The total number of events in the ESG approach has 249 events, while the Statechart approach has only 164 events.

After evaluating the number of generated test cases and events for Statechart and ESG approaches, we investigated both of the approaches based on their transition pair coverage or in other words event pair coverage. In this larger study, we have observed that with the Statechart approach 767 test cases are generated with 3980 events. On the other hand, with the ESG approach, we were able to generate one test case, with 109 events, which has reduced the test generation cost on a vast scale.

Nevertheless, we compared the negative tests for both approaches. For the Statechart approach, negative test case generation was not available, while we were able to generate test cases for ESG. We recall that the transitions in an Statechart are equivalent to the event nodes in the corresponding ESG. Therefore, we can easily define and generate invalid event relationships, by adding arcs between events to the ESG for negative testing. With the ESG approach, we generated 2768 test cases with 11678 events.

C. Threats to Validity

A threat to validity is that our evaluations are only based on a single large SPL, which is the Smart Home SPL. Therefore, the evaluations outlined above may not be representative or cannot be generalized. ESGs might behave differently on different SPLs and designs. Even though there is a large dataset of SPL [9] with feature models, there are very few Statechart diagrams provided for the features of a feature model. Therefore, to address this problem we tried to find case studies that used Statechart and SPL together. Best to our knowledge we were only able to find one [5], which we included in our case study, and create an Statechart for a larger SPL (Smart Home) to extend our case study.

The test case generation problem can be considered as an optimization problem as mentioned in Section III-D. Therefore, test generation with Statechart diagrams can be tackled as a traveling salesman problem (NP-hard) for state coverage and can be tackled as a Chinese postman problem (NP-complete) for transition coverage. These optimization problems are well studied, and there are various heuristic and meta-heuristic approaches to solve these problems that at least find local optimum solutions. Thereby, different automated model-based testing tools might generate a different number of test cases with a different number of events for the Statechart, which could be another threat to this study. For instance, Weißleder et al. [5] generated test cases both manually and with an automated model-based testing tool for the same Statechart. Their manually generated test cases and events were fewer than the automated tool. However, it is important to recall that manually generating test cases may not be feasible for larger and complex systems. Thereby, we used state-of-the-art tool (Conformiq) that has been used in literature for generating test cases in for Statechart, and to show that it is not feasible on a larger system, we ran Conformiq on our own Smart Home Statechart diagram.

V. Related Work

Different models are proposed for SPLs to enable explicit mapping features to behavior. One proposal is featured transition systems [24], which are transition systems like a finite state machine, where each transition is labeled with a feature expression, commonly a Boolean expression based on the FM of the SPL. This enables to specify the variants that execute this transition. Sampath, Bryce, and Memon suggest to combine multiple criteria into a hybrid, based on behavioral modeling [25]. Apart from featured transition systems other techniques based on modal transition systems with variability/constraints [26], or featured-modal contract automata [27], product lines process algebra [28] can be used for behavioral modeling.

A method introduced by Lackner, Schmidt, and Weißleder links feature models with Statecharts to consider the concept of variability binding, and thus, to consider also the direction of the test execution, that is, top-down or bottom-up [5], [15]–[17]. None of these studies have been considered generating negative tests, which models faults. This is one main point that differentiates our study from those works. Another main point of differentiation is that the focus in our approach is on events rather than states since states' controllability and observability may be more complicated than events in behavior-based testing of SPLs.

Model-based testing (MBT) of SPLs is not limited to the 150% finite state machine models technique. ScenTED (Scenario-based TEst case Derivations) [29] and CADeT (Customizable Activity diagrams, Decision tables, and Test specifications) [9], [30] are other behavior-based modeling approaches with their test generation aiming only to positive tests.

The above approaches do not consider feature-based specification. Feature-based analysis of SPLs including testing is built on feature-based specification [31]. Feature-based analysis can only detect issues within a certain feature, which is very important in domain engineering. Since features are only analyzed in isolation, feature-based analysis cannot reason about issues across features [31], which is required in application engineering. Several features work as expected in isolation but lead to unexpected behavior in combination [32]. ESGs can be used to model features and generate tests for them. Since ESGs are composable, feature ESGs can be put together for product ESGs as exemplified in this paper as well as in [33]. Their composability and their ability for holistic testing, covering both positive and negative tests, make ESGs favorable for MBT of SPLs.

TABLE III: Smart Home test case and number of events

Test Type	Positive/Negative Test	Coverage Type	Statechart Approach		ESG Approach	
			#test cases	*#events*	*#test cases*	*#events*
Bottom-Up Testing	Positive Test	Trans./Event Cov.	42	164	33	249
		Trans./Event Pair Cov.	767	3980	1	109
	Negative Test	-	n/a	n/a	2767	11677

VI. Conclusion

This paper suggests an approach for testing SPL and its derivable variants by unifying variability and behavior modeling of the SPL under consideration.

The suggested model-based holistic testing combines positive testing to validate the requirements of how the system is supposed to behave in regular situations and negative testing how in irregular situations. Examples of the fault prototypes, listed in Section IV-A, explain the importance of negative testing for the practice.

The suggested strategy enables, based on variability binding mechanism [5], a bottom-up test to validate the SPL, and a top-down test to validate the variants and groups of variants. The test adequacy will be ensured based on the coverage of variability features and behavioral events and event sequences.

A simplified online shop was used as a running example to explain and illuminate the approach. This example and a larger, real-life-like smart home application were used for a comparison with a similar SPL testing technique. Comparisons confirm that bottom-up testing causes fewer costs, and the holistic testing has the advantage of enabling negative testing.

The scalability by adjusting the length of event sequences (dependent on the test budget) that are to be covered, and efficiency are further advantages of the approach, ensured by heuristic test generation algorithms borrowed from graph theory and supported by a dedicated test tool. Higher costs of the initial test step to cover event sequences, and costs to learn the method and its tool are on the disadvantage side of the approach.

To sum up, the paper answered the initial questions asked in Section I: All variability points and events/event sequences that form the structure and the behavior of the SPL can efficiently be covered, considering also their interaction with the user and environment.

Our future research includes developing a formal, stringent model of the introduced heterogeneous modeling and the related techniques, including the top-down and bottom-up testing. Further work is necessary also to form a framework for enabling real tests to validate SPL and its variants.

References

[1] F. Belli, "Assuring dependability of software reuse: An industrial standard," in *International Conference on Software Technologies*, 2013, pp. 72–83.

[2] "IEC/PAS 62814 Dependability of Software Products Containing Reusable Components – Guidance for Functionality and Tests," 2012.

[3] F. Belli and F. Quella, *A Holistic View of Software and Hardware Reuse*. Springer Science and Business Media LLC, 2021.

[4] K. Pohl, G. Böckle, and F. J. van Der Linden, *Software product line engineering: foundations, principles and techniques*. Springer Science & Business Media, 2005.

[5] S. Weißleder and H. Lackner, "Top-down and bottom-up approach for model-based testing of product lines," *arXiv preprint arXiv:1303.1011*, 2013.

[6] H. Grönniger, H. Krahn, C. Pinkernell, and B. Rumpe, "Modeling variants of automotive systems using views," *Modellbasierte Entwicklung von eingebetteten Fahrzeugfunktionen (MBEFF), Informatik Bericht*, vol. 1, pp. 76–89, 2008.

[7] C. Seidl, D. Wille, and I. Schaefer, "Software Reuse: From Cloned Variants to Managed Software Product Lines," in *Automotive Systems and Software Engineering*. Springer, 2019, pp. 77–108.

[8] H. Cichos, S. Oster, M. Lochau, and A. Schürr, "Model-based coverage-driven test suite generation for software product lines," in *International Conference on Model Driven Engineering Languages and Systems*. Springer, 2011, pp. 425–439.

[9] J. Myhill, "Finite automata and the representation of events," *WADD Technical Report*, vol. 57, pp. 112–137, 1957.

[10] F. Belli, "Finite state testing and analysis of graphical user interfaces," in *IEEE International Symposium on Software Reliability Engineering*, 2001, pp. 34–43.

[11] F. Belli, C. J. Budnik, and L. White, "Event-based modelling, analysis and testing of user interactions: approach and case study," *Software Testing, Verification and Reliability*, vol. 16, no. 1, pp. 3–32, 2006, publisher: Wiley Online Library.

[12] F. Belli and C. J. Budnik, "Test minimization for human-computer interaction," *Applied Intelligence*, vol. 26, no. 2, pp. 161–174, 2007.

[13] A. V. Aho, A. T. Dahbura, D. Lee, and M. U. Uyar, "An optimization technique for protocol conformance test generation based on UIO sequences and rural Chinese postman tours," *IEEE Transactions on Communications*, vol. 39, no. 11, pp. 1604–1615, 1991.

[14] K. El-Fakih, R. M. Hierons, and U. c. Turker, "$\mathcal{K}$-branching uio sequences for partially specified observable non-deterministic fsms," *IEEE Transactions on Software Engineering*, pp. 1–1, 2019.

[15] D.-I. H. Lackner, "Domain-Centered Product Line Testing," 2016, publisher: Humboldt-Universität zu Berlin.

[16] H. Lackner and M. Schmidt, "Potential Errors and Test Assessment in Software Product Line Engineering," *arXiv preprint arXiv:1504.02443*, 2015.

[17] H. Lackner and B.-H. Schlingloff, "Advances in testing software product lines," in *Advances in computers*. Elsevier, 2017, vol. 107, pp. 157–217.

[18] ESG Test Suite Designer. [Online]. Available: http://download.ivknet.de/

[19] S. Weißleder, F. Wartenberg, and H. Lackner, "Automated test design for boundaries of product line variants," in *International Conference on Testing Software and Systems*, 2015, pp. 86–101.

[20] "Conformiq Designer." [Online]. Available: https://www.conformiq.com/products/conformiq-designer/

[21] F. Belli and A. Hollmann, "Test generation and minimization with" basic" statecharts," in *Proceedings of ACM Symposium on Applied Computing*, 2008, pp. 718–723.

[22] SPLOT Research. [Online]. Available: http://splot-research.org

[23] Feature IDE. [Online]. Available: https://featureide.github.io

[24] A. Classen, M. Cordy, P.-Y. Schobbens, P. Heymans, A. Legay, and J.-F. Raskin, "Featured transition systems: Foundations for verifying variability-intensive systems and their application to ltl model checking," *IEEE Transactions on Software Engineering*, vol. 39, no. 8, pp. 1069–1089, 2012.

[25] S. Sampath, R. Bryce, and A. M. Memon, "A uniform representation of hybrid criteria for regression testing," *IEEE Transactions on Software Engineering*, vol. 39, no. 10, pp. 1326–1344, 2013.

[26] M. H. ter Beek, A. Fantechi, S. Gnesi, and F. Mazzanti, "Modelling and analysing variability in product families: model checking of modal transition systems with variability constraints," *Journal of Logical and Algebraic Methods in Programming*, vol. 85, no. 2, pp. 287–315, 2016.

[27] D. Basile, M. H. ter Beek, P. Degano, A. Legay, G.-L. Ferrari, S. Gnesi, and F. Di Giandomenico, "Controller synthesis of service contracts with

variability," *Science of Computer Programming*, vol. 187, p. 102344, 2020.
[28] A. Gruler, M. Leucker, and K. Scheidemann, "Modeling and model checking software product lines," in *International Conference on Formal Methods for Open Object-Based Distributed Systems*, 2008, pp. 113–131.
[29] A. Reuys, E. Kamsties, K. Pohl, and S. Reis, "Model-based system testing of software product families," in *International Conference on Advanced Information Systems Engineering*, 2005, pp. 519–534.
[30] E. M. Olimpiew and H. Gomaa, "Reusable model-based testing," in *International Conference on Software Reuse*. Springer, 2009, pp. 76–85.
[31] T. Thüm, S. Apel, C. Kästner, I. Schaefer, and G. Saake, "A classification and survey of analysis strategies for software product lines," *ACM Computing Surveys*, vol. 47, no. 1, pp. 1–45, 2014.
[32] M. Calder, M. Kolberg, E. H. Magill, and S. Reiff-Marganiec, "Feature interaction: a critical review and considered forecast," *Computer Networks*, vol. 41, no. 1, pp. 115–141, 2003, publisher: Elsevier.
[33] T. Tuglular, M. Beyazıt, and D. Öztürk, "Featured event sequence graphs for model-based incremental testing of software product lines," in *IEEE Computer Software and Applications Conference*, vol. 1, 2019, pp. 197–202.

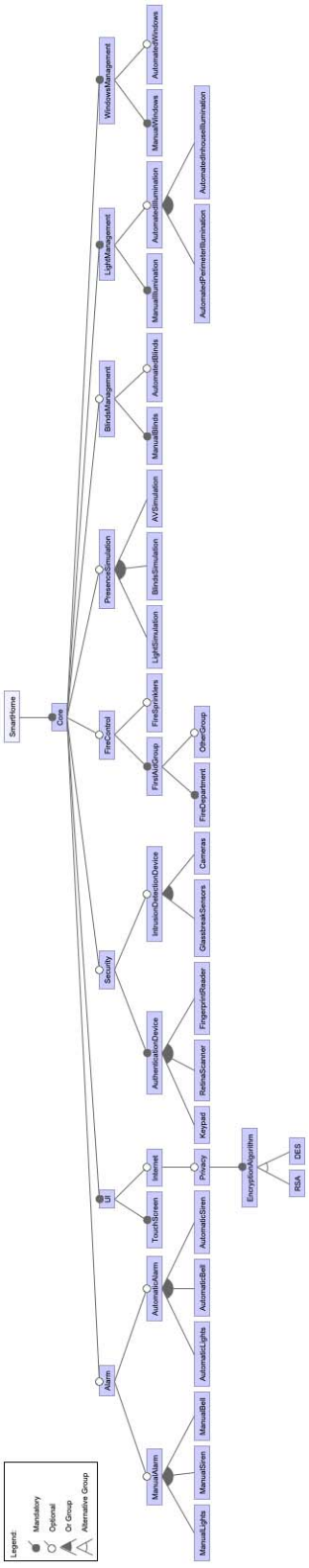

Fig. 10: Smart Home Feature Model

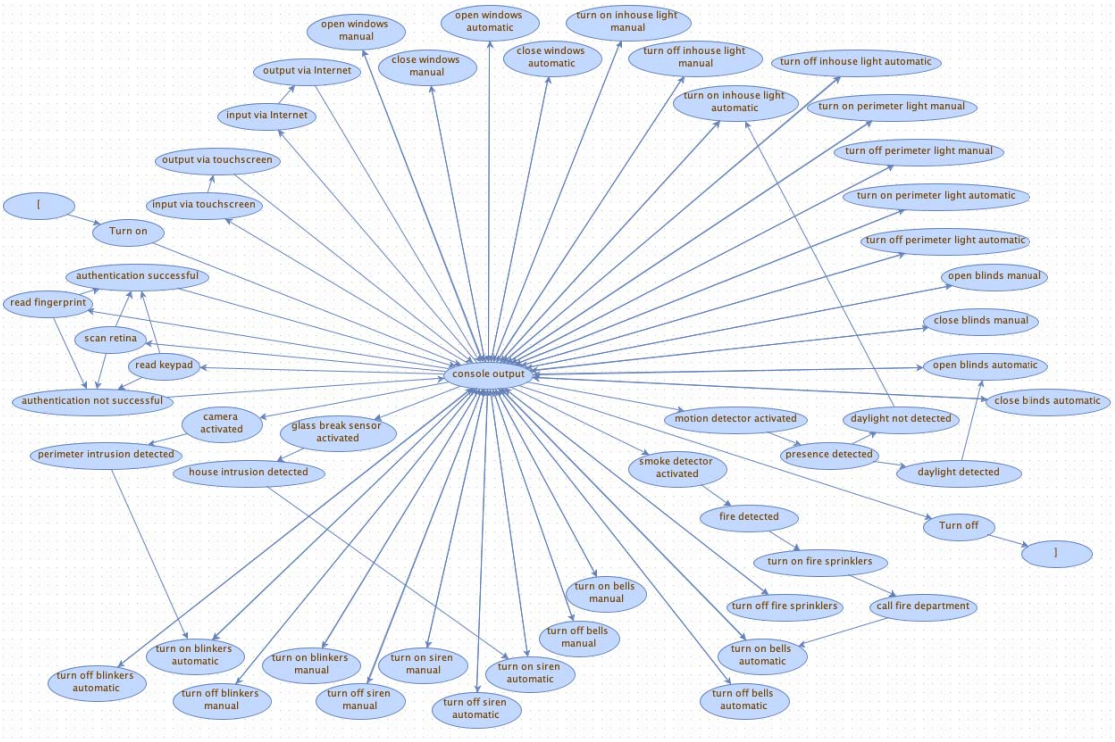

Fig. 11: ESG of Smart Home

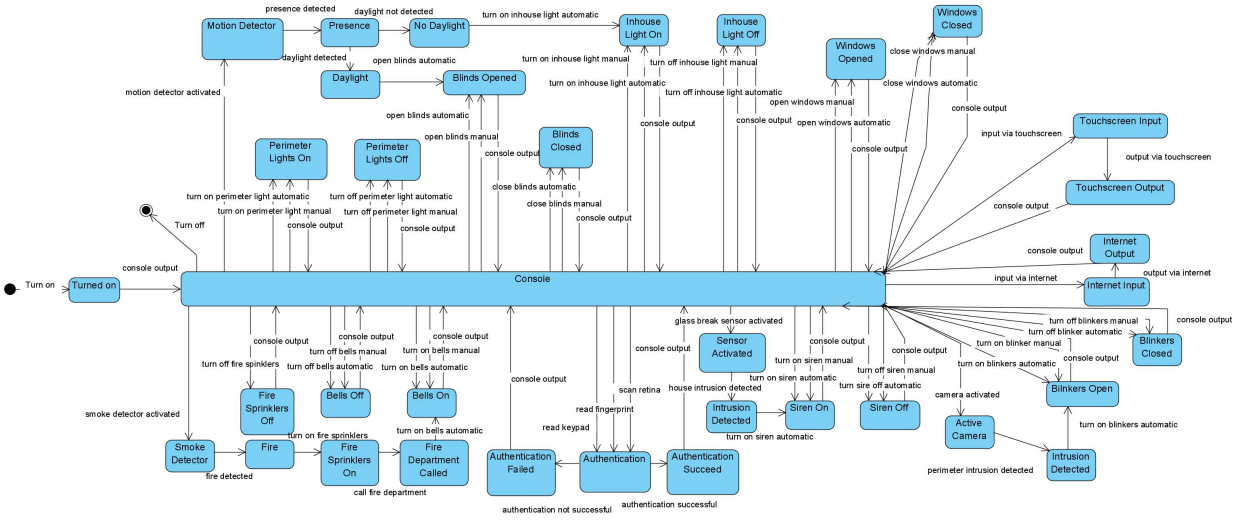

Fig. 12: Statechart Diagram of Smart Home