

Using Scope Scenarios to Verify Multiple Variability Models

Matias Pol'la, Agustina Buccella(✉), and Alejandra Cechich

GIISCo Research Group, Faculty of Informatics,
UNComa University, Neuquen, Argentina
agustina.bucella@fi.uncoma.edu.ar

Abstract. Analyzing model variability represents a rapidly evolving discipline with increasing applications in different fields. Several efforts have addressed the analysis of a particular variability model represented, for instance, as feature models (FM). However, due to the proliferation of interrelated models, a major challenge today is detecting inter-model inconsistencies; that is, analyzing inconsistencies among inter-related variability models. In this paper, we introduce a proposal for verifying multiple variability models by using scope scenarios. Our approach is based on the SeVaTax method for building variability through functional datasheets, which are inputs to the process. Preliminary evaluation shows promissory results in terms of detected inconsistencies; however performance rises as a challenging issue for spreading the findings.

Keywords: Variability analysis · Scope scenarios · Variability models · Validation

1 Introduction

The product line approach provides a systematic way for reusing software assets. One of its current challenges is to model the variability between the core assets and the applications. Variability is modeled in several ways, such as attached information in UML diagrams [14], variation points that allow a developer to extend components at these points, or inside functional datasheets, as we have proposed previously [5,6]. In this last case, our proposal –SeVaTax– builds variability models upon the work in [15], in order to facilitate automated analysis.

In [3] there is a broader literature review of automatic analysis of feature models (FM), based on a general process that defines a set of tasks for evaluating FM. The process takes as input one FM and translate it into a formal representation, which includes variability in logical terms. Then, a solver is responsible for validating the formal model generated, and also receives an scenario/query

This work is partially supported by the UNComa project 04/F009 "Reuso de Software orientado a Dominios - Parte II" part of the program "Desarrollo de Software Basado en Reuso - Parte II".

© Springer Nature Switzerland AG 2019
S. Misra et al. (Eds.): ICCSA 2019, LNCS 11623, pp. 383–399, 2019.
https://doi.org/10.1007/978-3-030-24308-1_32

set, which determines the results of the analysis process. These queries represent the set of questions that solvers are capable of answer. In this sense, in the literature there exists some common validation scenarios [3,9] such as *valid model*, *valid instantiation*, *all products*, etc. These scenarios are important in order to analyze the capabilities of the solvers.

In [15] we have introduced our SeVaTax process covering all activities of the automated variability analysis as presented in [3]. Our approach towards variability analysis lets the developer/reuser build his/her verification model including diverse types of analysis through the use of scenarios. Particularly, scenarios can address more complex situations, such as managing multiple configurations of the same variability model, or checking consistency of multiple but related variability models. Thus, our contribution in this paper focuses on analyzing variability in multiple models by using scope scenarios. The analysis of this type of scenarios allows us to identify inconsistencies that can emerge from the use of multiple models and variabilities, and to provide a pre-defined set of answers to repair them. At the same time, as the analysis of multiple models generates more resource consumption, we analyze the performance our tool.

This article is organized as follows. Next section describes related work in the literature. Then, we briefly describe our SeVaTax process together with its main components and activities. Following, we focus on the scope operators and the validation scenarios designed for detecting and repairing inconsistencies in multiple models. A performance analysis is also included. Conclusions and future work are discussed afterwards.

2 Related Work

Several literature reviews have been presented analyzing the specific activities involved in a variability analysis process [3,10,20]. For example, in [10] authors analyze domain analysis tools, by considering any modeling approach and evaluating functionalities supported by the tools (according to modeling, derivation and validation). The reviews considering works closer to our proposal are presented by Benavides et al. [3] and Sree-Kumar et al. [20]. They analyze the variability analysis processes taking into account specific techniques, methods and/or resources involved. However, these reviews are only focused on FM, i.e. they do not consider other modeling approaches. This is an important aspect because the modeling approach determines the way the activities of the variability analysis can be applied; for example, the type of inconsistencies that can be generated, the capabilities that reasoners/solvers can provide, etc. Thus, in Table 1 we summarize some proposals in the literature focused on the variability analysis process. We include in the table proposals which vary in the modeling approach in order to show the formal language, the reasoner/solver applied and the scenarios validated. In this case we include three general scenarios, redundancies (R), anomalies (A), and inconsistencies (I) [9].

For example, in the table we can see the FAMA-FM and FAMA-OVM proposals whose variability analysis processes translate FM and OVM models into CSP

Table 1. Summary of automated analysis approaches

Approach	V.M.	F.M.	Solver	Scenarios		
				R	A	I
FAMA-FM [19]	FM	CSP - BDD	SAT4j[a] - JaCoP GPL[b] -	x	x	x
			Choco[c] - JavaBDD[d]			
FAMA-OVM [17]	OVM	DL - PL	Choco - JavaBDD -			x
			Sat4j - JaCoP GPL			
Metzger et al. [13]	OVM - FD	Own	SAT4j			x
Rincon et al. [16]	FM	OWL	JESS[e]		x	x
VariaMos [11]	Independent	Own	SWI-Prolog[f]	x	x	x
S.P.L.O.T. [12]	FM	3-CNF	JavaBDD		x	x
CLAFER [2]	FM	OWL	Alloy Analyzer[g]			x
DOPLER [7]	Own	PL	JBoss Drools[h]	x	x	x
FAMILIAR [1]	Own	PL	JavaBDD - Sat4j			x
Zaid et al [22]	FM	OWL	Pellet[i]			x

[a] http://www.sat4j.org/; [b] http://www.lth.se/jacop/;
[c] www.choco-solver.org; [d] http://javabdd.sourceforge.net/;
[e] https://www.jessrules.com/links/;
[f] http://www.swi-prolog.org/; [g] http://alloytools.org;
[h] https://www.drools.org/; [i] http://pellet.owldl.com/;

and BDD, and DL (Description Login) and PL (Propositional Logic), respectively. In addition they use four reasoners (SAT4j, JaCoP GPL, Choco, and JAvaBDD) in order to find possible redundancies, anomalies and inconsistencies. In the case of FAMA-OVM, authors only define mechanisms for inconsistency detection. Another example is VariaMos that provides its own variability modeling approach to be analyzed by a SWI-Prolog reasoner. It also generate mechanisms for the three scenarios. Finally we can cite the work of Metzger et al. [13] in which authors use two modeling approaches, OVM and FD (feature diagram) for documenting different artifacts, and apply a SAT solver for detecting inconsistencies among them.

There are also other works, more related to the modeling strategies of different variability diagrams during a software product line development, such as [8,18]. Both works propose the use of more than one FM, for documenting or developing multiple product lines, which must be merged or integrated in some stage of the development. However, these proposals do not provide explicit mechanisms for identifying or repairing inconsistencies or anomalies during the process.

The FAMA-OVM proposal is the closest to our work due to OVM is based on the annotation of variability over software artifacts, such as activity or collaboration diagrams of UML. This particularity makes that the variability can be defined in different points, which adds a higher level of complexity when the

models must be analyzed together. However, this issue is not addressed in [17]; thus, in this work we propose an extension of our SeVaTax process in order to face to the complexity of analyzing variability belonging to multiple models.

3 Automated Analysis of Variability Models: The SeVaTax Process and Tool

In [15] we describe our SeVaTax tool based on the SeVaTax analysis process. In Fig. 1 we can see the software components and inputs/outputs included in the tool. These components are responsible for performing the following activities:

- The *translator* component (2) takes the variability models (represented as JSON files) and translates them to a formal representation. These models, named functional datasheets [5,6] (1), are described next in Module 1. This component generates two different representations of each variability model, a CNF representation (detailed in Module 2), and a set of underlying structures representing different aspects associated to each service (detailed in Module 2). These structures were developed together with a set of functions (methods) in Angular1, which support the identification of the validation scenarios described in Module 4.
- The *validator* component (3) takes as input the CNF representation and the underlying structures defined in the previous component and performs two main verifications. Firstly, it evaluates each variability model (datasheet) separately; and secondly evaluates the complete set of models (SPL Platform). This validator, also implemented in Java, uses a SAT solver (Sat4j) together with specific functions for supporting the validation scenarios. In Module 3 we describe some of these functions in detail.
- The *configurator* (5) component takes the validated variability models and provides an environment for instantiating the variability (creating a new product of the line). Following, this instantiation is again validated (by the *validator*).
- The *result viewer* (4) component, reports the anomalies or problems found from analyzing the variability models, and performs some corrections (if it is possible). Furthermore, this component allows to modify the variability models in those anomalies without an automatic solution. So, it is an iterative process in which the outputs are the datasheets modified. In Module 4 we describe some of these functions in detail.

The analysis process is divided into five modules, which use the components previously introduced as follows.

3.1 Module 1: The SeVaTax Variability Model

The *SeVaTax variability model* [5,6] is based on variability annotation of collaboration diagrams (of UML). The required variability, according to the functionality to be represented, can be attached to the diagrams by using the OVM

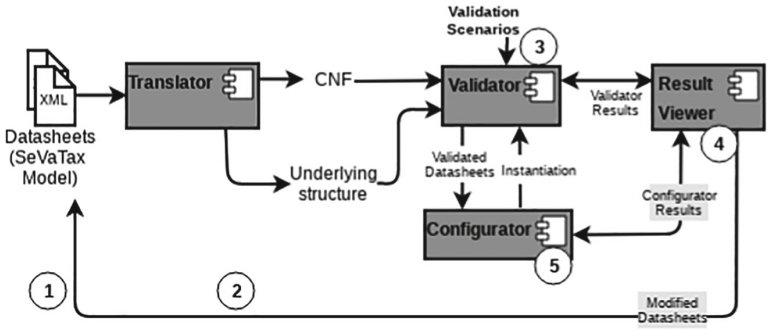


Fig. 1. SeVaTax analysis process according to the general analysis process

notation according to the interactions defined in Table 2. We divide these interactions into *variability types*, for the four first rows; *dependencies* for the fifth, sixth and seventh rows; and *scope operators* for the two last ones.

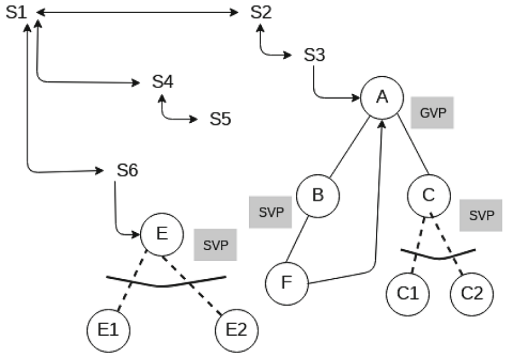
Three different interactions, unknown for other proposals, are the *use* dependency and the two *scope operators*. The first one specifies a dependence between common services, which are not necessarily associated with a variation point. The other, is related to the *scope* defined for a variation point when it must be instantiated [4]. A *Global Variation Point* specifies that if the variation point is instantiated in a specific way, it will be applied in that way for all functionality including that variation point; and a *Specific Variation Point* specifies that the instantiation of the variation point is particular for each functionality including that variation point.

Table 2. Interactions defined for modeling functional datasheets

Dependency	JSON property	Graphical Notation
Mandatory variation point	*MandatoryVP*	
Optional variation point	*OptionalVP*	
Alternative variation point	*AlternativeVP*	
Variant variation point	*VariantVP*	
Use	*use*	
Requires	*requires*	
Excludes	*excludes*	
Global variation point	*GlobalVariationPoint*	GVP
Specific variation point	*SpecificVariationPoint*	SVP

As in this article we are focused on the *scope operators when multiple models are involved*, in Fig. 2(a) and (c) we show two datasheets with Specific VPs and

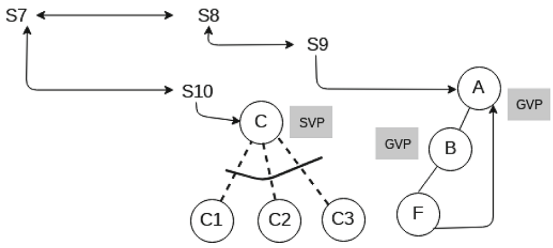
Global VPs. In both figures we can see a set of interactions of services in which we name S1...Sn to those services that are not involved in variation points, and A, B, C, etc. to those involved in variability dependencies. In Datasheet 1 we modeled six fixed services (from S1 to S6) and three variation points (A, B and C). Each of them is labeled with a scope operator, in which A is a Global VP, and B and C are Specific VPs. Also, the A-B-F subtree is related by mandatory variabilities, and C1 and C2 are variants included in an alternative variation point. The same happens with E that is a Specific VP involving two alternative variants. Following in Datasheet 2 we have four services without variability (S7–S10) and three variation points (A, B, C). Two of them (A and B) are Global VPs (as in datasheet 1), and the other (C) is an Specific VP involving three alternative variants (C1–C3).



a) Datasheet 1

```
{
"Datasheet1": {
    "id": "Model_Name",
    "service": {
      "..."
"name": "S3",
"use":{
"Service":
 {"name": "A",
  "GlobalVariationPoint": {
  "MandatoryVP": {
    "service": [
     { "name": "B", ..},
     { "name": "C"
        "SpecificVariationPoint": {
        "AlternativeVP":{
           "service": [
             { "name": "C1",
             {"name": "C2"}
             ]
....}
```

b) JSON file for representing Datasheet 1



c) Datasheet 2

Fig. 2. (a) Datasheet 1; (b) JSON file for the Datasheet 1; (c) Datasheet 2

Then, these datasheets are translated to JSON files by using the tags defined in Table 2. In Fig. 2(b) we show an extract of the datasheet 1 in which we show the S3 service with a use dependency to the variation point A, which is defined as a GlobalVP. This variation point has the services B and C with a mandatory

interaction, and then C defined as a SpecificVP with alternative variants (C1 and C2)[1].

3.2 Module 2: Translator and Formal Representation

The *translator* is responsible for parsing the JSON files (from the datasheets) and generating two outputs, a formal representation (in CNF) and an instantiation of the underlying structures (Module 2 of Fig. 1). The CNF representation was presented in [15] for each interaction defined in Table 2.

Underlying structures are defined for identifying the different scenarios. In Fig. 3(a), we show these structures instantiated according to the datasheet 1 (Fig. 2a). The structure is composed of a set of objects (white boxes) with different attributes, for instance, object lists (gray boxes). Thus, each datasheet is represented by a name and a list of services, and each service by a name. In addition, a service can contain a list of variation points and a list of dependencies. To represent the variation points, we register the variation points scope (which is unique) together with the variation points list. Each variation point contains a variability type and a variant list. Each variant corresponds to a service. Regarding to the dependency list, each dependency is composed of a dependency type (Include, Exclude or Uses) and the target service. Each service belongs to one or more datasheets. In this way, it is possible to represent cross-dependencies.

By following with the datasheet 1 example of Fig. 2, in Fig. 3(b) we can see the CNF translation of the JSON file, in which we have 15 literals (corresponding to the 15 services) composed of 28 clauses. We show 5 of these clauses in which we can see that clause 3 denotes that *S3 requires A*, clause 4 denotes that *A requires B*, clause 5 denotes that *A requires C*, and clauses 6 and 7 specify that the alternative variation point C is represented by two clauses denoting the XOR logic operator (*C requires (C1 XOR C2)*).

3.3 Module 3: Validator

With respect to the *solver*, in [15] we have described the way a set of 17 validation scenarios are defined into five categories. The three main categories are *general*, *scope* and *cross dependency*. The first category is then subdivided into *model error*, *dead service* and *redundancy*.

In the case of the *scope scenarios*, here we extend the categories in a more exhaustive list of possible problems found. Thus, the new list of scenarios to be validated are:

– *False SpecificVP (F-SVP):* when a Specific VP is related to an only one variant service depending on a Global VP. Then, the scope will be always GlobalVP, because, independently on the way the service is configured, the VP will be always Global. The next variability model denotes a F-SVP:

[1] This translation of the datasheets to the JSON file is still manual.

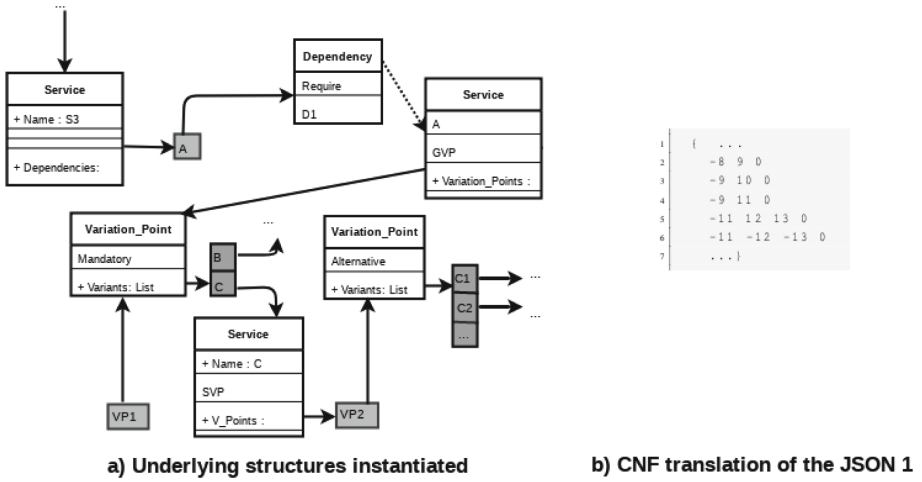
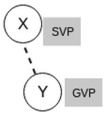


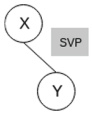
Fig. 3. Translations of datasheet 1



This scenario is validated by using the *Services_Consume_Variation_Point(VP)* function, which given a variation point $'VP'$ returns the number of services that use it; and when this number is one, we have a F-SVP:

```
Services_Consume_Variation_Point(VP):
  For all S in (Dependencies_List U Variant_List)
    If ( VP==S )Count ++
    If Count <= 1
       return F-SVP
```

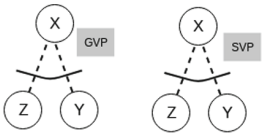
– *Mandatory SpecificVP (M-SVP)*: when a Specific VP is conformed only by mandatory variant points. Then it can never contain different configurations. The next variability model denotes a M-SVP:



To evaluate this scenario we use the *Mandatories_Childs(VP)* function, which given a variation point $'VP'$ returns true if all its variants are mandatory.

```
Mandatories_Childs(VP):
  For each VP in VariationPoin_List
    If  VP.VT is mandatory?
       Return M-SVP
```

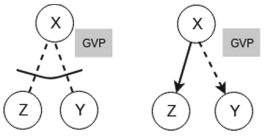
– *Contradictory Scopes (C-S)*: when the same VP with exactly the same variability relationships presents different scopes in different datasheets. The next variability model denotes a C-S:



To evaluate this scenario, for each service that represents a variation point, we register its scope into the underlying structure. Therefore, when registering the scope, we check if the service has the same scope. In all other cases, we are in the presence of a scope inconsistency.

```
Service_Scope(S):
  For each S1 in Datasheets_List
     If  (S1.name == S.name and S1.scope != s.name)
       Return C-S
```

– *False GlobalVP (F-GVP)*: when the same VP with different variability relationships is defined as a GlobalVP. Then, this can never be possible because in GlobalVPs the configurations must be the same. The next variability models of different datasheets denote a F-GVP:



To evaluate this scenario we use the underlying structures analyzing each GlobalVP. Then, we analyze the set of datasheets comparing their variability relationships and detect those situations in which the structures are different.

```
Global_analisys(VP):
   For each VP in VariationPoin_List
     For each Datasheet
       For each VP_D in Datasheet
         If VP == VP_D?
           If VP.relations != VP_D.relations ?
                Return F-GVP
```

Each of these scenarios must be identified within the different datasheets in order to generate corrections. That is, depending on the scenario, we can generate specific results as we will explain in the following module (module 4). For our two datasheets of Fig. 2, we can identify the scenarios described before that are labeled in Fig. 6 (Fig. 4).

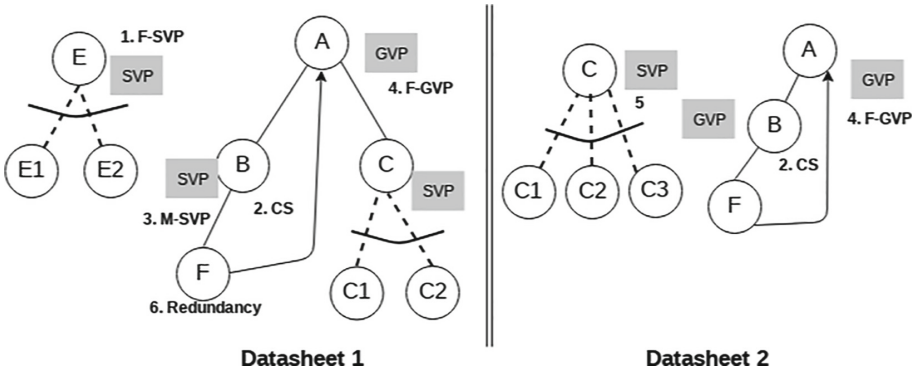


Fig. 4. Identified scenarios in datasheets 1 and 2

For example in (4) we identified a F-GVP because the variation point A is defined with different variability relationships (in datasheet 1, it has two mandatory services; and in datasheet 2, it has only one), so it can be only SpecificVP. It necessarily has different ways to instantiate them. Then, in (2) we identified a C-S scenario because the same variation point has two different scopes in the two datasheets. A different case is (6), in which we add a redundancy only to show that these scenarios, evaluated in [15] are also identified. In this case is redundant because the require dependency is unnecessary. Finally, in (5) the tool does not identify any inconsistency because the two variation points C in both datasheets are correct SpecificVP; and configurations will be different.

3.4 Module 4: Result Viewer

In order to provide more effective solutions when analyzing variability models, the SevTax tool classifies results into a pre-defined set of answer types, such as warnings, identifications, and repairs. In the case of scope scenarios, we can always fix the problem to repair the models. Table 3 shows the action performed once each scenario is detected.

Then, Fig. 7 shows the datasheets with the reparation actions performed. As we can see, scenario (1) was modified to GlobalVP. For the case of (2) and (3) scenarios, involved into the same variation point B, the tool first analyzes the C-S scenario (2) modifying the variation point to SpecificVP. In this point, the variation point B in both datasheets will be SpecificVP. After that, the tools analyzes the M-SVP (3) scenario, modifying B to GlobalVP in both cases.

Table 3. Scope validation scenarios and answer type

Scenario	Answers	Action
False SpecificVP (F-SVP)	R	Modify To GlobalVP
Mandatory SpecificVP (M-SVP)	R	Modify To GlobalVP
Contradictory scopes (C-S)	R	Modify To SpecificVP
False GlobalVP (F-GVP)	R	Modify To SpecificVP

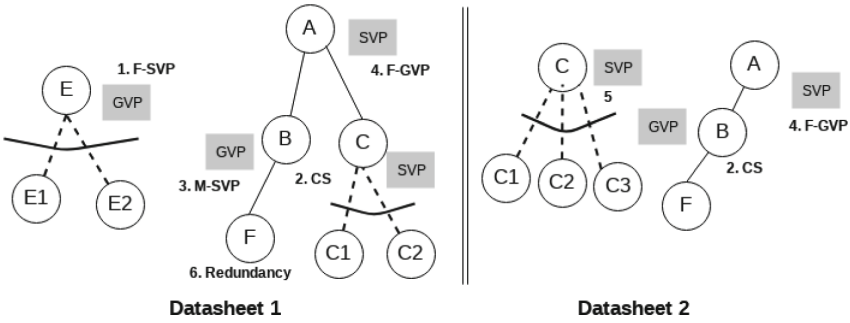


Fig. 5. Datasheets 1 and 2 corrected after validating them

Thus, the final reparation will have this configuration. It is important to highlight that if the process is executed in inverse order, the result will be the same. Finally, in (6), as it is a redundancy, the tool deletes the requires dependency.

3.5 Module 5: Configurator

The *configurator* allows developers to build a specific product by means of instantiating the functional datasheets included into an SPL platform. Therefore, this component presents a user interface (UI) that allows developers to select the different variant services (associated with variation points). To perform that, a developer receives as input the validated functional datasheets (without anomalies), instantiates them, and generates a new set of datasheets without variability. This new set must be again evaluated (by Module 4) due to it can also contain inconsistencies generated by selecting/deselecting variants.

Figure 6 shows the configuration of datasheets 1 and 2 after validating them. So, the configuration is made over the datasheets showed in Fig. 5. In this case, the alternative variation point E contains an inconsistency due to both variants E1 and E2 have been selected. This inconsistency is then detected internally by using the SAT solver of the tool.

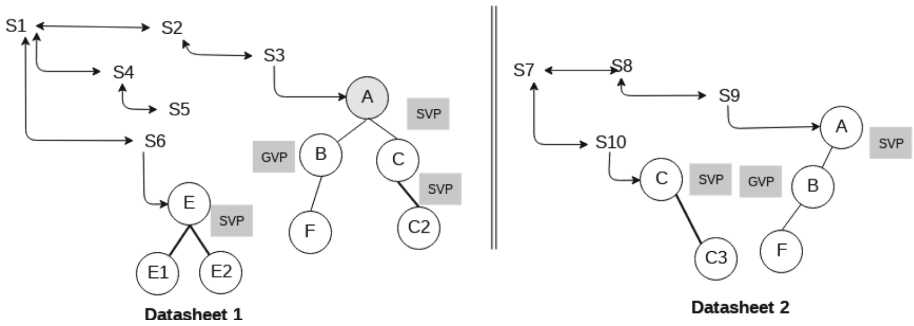


Fig. 6. One possible instantiation of Datasheets 1 and 2

4 Evaluation

The SevaTax process has been designed and implemented as a web tool according the components defined in Fig. 1. The tool is based on the Angular framework and it is available in github at the following link: https://github.com/IngSisFAI/sevataxtool.git. At the same time, a first prototype of the tool is also available at: http://analysis.fi.uncoma.edu.ar. Based on this tool, in [15] we performed a tool analysis, comparing our SeVaTax process to S.P.L.O.T and VariaMos. However, this type of comparison is not useful here due to the scope scenarios are not supported by these tools. This is because most of the tools are based on FM variability models, and those that are based on OVM are not considering this type of problems. At the same time, and considering that the complexity of analyzing multiple model is more resource-consuming, we make a performance analysis according to guidelines from [21]. Following we describe the experiments and the generated results.

A. Selection of Variability Models or Test Cases

We designed and implemented a tool that performs the automatic datasheet generation, taking into account the interactions defined for modeling datasheets presented in Table 3. Thus, the output of the tool is a set of datasheets (or variability models) in compliance with the SeVaTax models in JSON format. That is, this tool uses the model restrictions defined in the SeVaTax Variability Model for generating aleatory models (in datasheets) together with a wide set of validation scenarios included in them. Thus, the tool generated 15 datasheets with 250 services and 85 variation points (each variation point has attached a GlobalVP or SpecificVP).

B. Evaluation Design

For this experiment we define the following hypotheses:

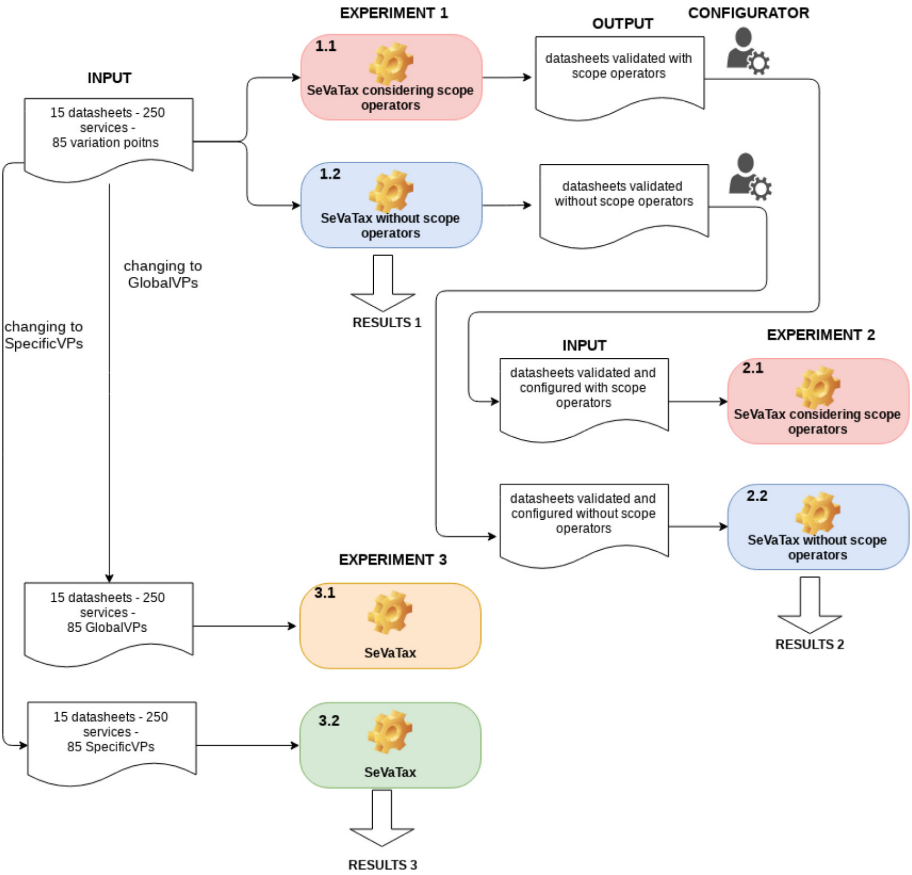


Fig. 7. Experiment design (Color figure online)

Hypothesis 1 (H1): *How does the execution time of the datasheet validation scale-up as the number and type of scope scenarios ?*

Hypothesis 2 (H2): *How does the execution time vary when derived models are involved?*

The hypotheses intent to analyze the way our tool behaves when the number of scenarios and inconsistencies increase, and when the scenarios were or not instantiated. The experiment was executed in a Notebook with an Intel i5 8th Generation and 4 GB DDR2 RAM.

For validating these hypotheses, we used the datasheets generated (as specified in A), but changing the analysis performed over them. Thus we performed three type of experiments (showed in Fig. 7) for analyzing the execution times (in miliseconds):

- *Experiment 1:* We performed two different executions of SeVaTax: (1.1) taking into account the scope operators attached to the variation points (red box), and (1.2) without considering the scope operators (blue box). In this second test, the SeVaTax tool was changed in order to ignore the scope operators validations.
- *Experiment 2:* From the outputs of the 1.1 and 1.2 executions in which we obtained a modified set of datasheets (validated datasheets) we performed an aleatory configuration by using the SeVaTax tool generating again two new test cases: (2.1) datasheets validated and configured with scope operators and (2.2) datasheets validated and configured without scope operators. Here, we executed these two test cases.
- *Experiment 3:* We modify again the test cases generating and executing two new ones: (3.1) datasheets with only SpecificVPs (orange boxes), and (3.2) datasheets with only GlobalVPs (green boxes).

C. Evaluation Results

Figure 8 shows the results obtained from the execution of the three experiments. In the experiment 1 (Fig. 8(a)), we can see that the execution time is higher when scope operators are involved (red line shows the 1.1 results, and blue one 1.2). Then, Fig. 8(b) shows the results for the experiment 2 in which we can see that the execution times are now inverted because the test case with scope operators (2.1 - red points) generated a better result than the test case without them (datasheets without scopes - red points). In Fig. 8(c) we compared the execution times when the two scope operators are involved and we can observe that the execution time is worse when we have only GlobalVPs (3.1 - red line).

From these preliminary experiments, we can extract some highlights. Firstly, the tool has a worse performance when scope operators are involved. Also, the worst performance is obtained when we the number of GlobalVPs increases. This happens because a SAT solver cannot be applied in these cases, making mandatory to use and revise the underlying structures. In the case of GlovalVPs, these structures must be analyzed more than once making the process slower. That is, verifying GlobalVPs requires to analyze each variation point to maintain the same structure.

However, from experiment 1 and 2 we could observe that in spite of the time for validating datasheets with scope operators is longer than without them, after validation and configuration, the execution time decreases in comparison to the datasheets without scope. In Fig. 8(d) we can see that the reduction of found errors after configuration in datasheets with scope operators is lesser because some inconsistencies and redundancies have been already solved, specially when SpecificVPs are involved .

In this way, taking into account the two hypotheses, we evidenced that the execution times increase when scope scenarios are involved (Hypothesis 1) but these times are improved when the models were validated and configured (Hypothesis 2).

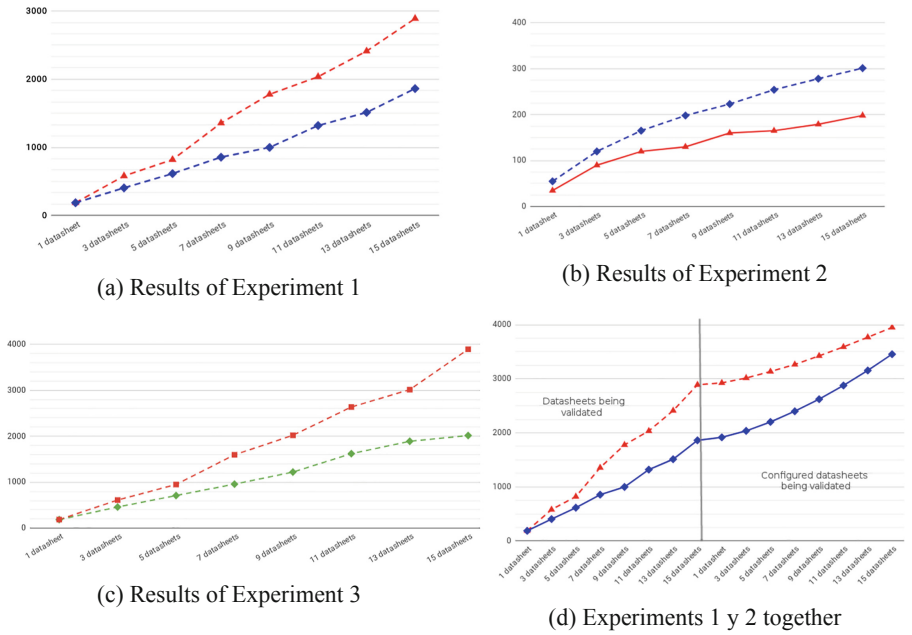


Fig. 8. Results of the execution of the three experiments (Color figure online)

5 Conclusion and Future Work

In this work we have focused on the analysis of multiple variability models and the inconsistencies that can occur among them.

In particular, we have focused on the fact that each software artifact contains several variability models attached, generating that other types of inconsistencies may be derived from incompatibilities among these models. In this article, our SeVaTax process has been extended with specific operators (called scope operators) together with a classification of the incompatibilities that we can found. Also, we have described the components and activities performed by our supporting tool in order to detect and even repair these inconsistencies.

As future work we are working on evaluating usability in order to analyze our supporting tool interface and compare it to accepted usability principles. Also, we are implementing a graphical interface for managing functional datasheets.

References

1. Acher, M., Collet, P., Lahire, P., France, R.: Familiar: a domain-specific language for large scale management of feature models. Sci. Comput. Program. **78**(6), 657–681 (2013). https://doi.org/10.1016/j.scico.2012.12.004
2. Bak, K., Diskin, Z., Antkiewicz, M., Czarnecki, K., Wkasowski, A.: Clafer: unifying class and feature modeling. Softw. Syst. Model. **15**(3), 811–845 (2016). https://doi.org/10.1007/s10270-014-0441-1

3. Benavides, D., Segura, S., Ruiz-Cortés, A.: Automated analysis of feature models 20 years later: a literature review. Inf. Syst. **35**(6), 615–636 (2010). https://doi.org/10.1016/j.is.2010.01.001
4. Brisaboa, N.R., Cortiñas, A., Luaces, M.R., Pol'la, M.: A reusable software architecture for geographic information systems based on software product line engineering. In: Bellatreche, L., Manolopoulos, Y. (eds.) MEDI 2015. LNCS, vol. 9344, pp. 320–331. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23781-7_26
5. Buccella, A., Cechich, A., Arias, M., Pol'la, M., Doldan, S., Morsan, E.: Towards systematic software reuse of GIS: insights from a case study. Comput. Geosci. **54**(0), 9–20 (2013). https://doi.org/10.1016/j.cageo.2012.11.014
6. Buccella, A., Cechich, A., Pol'la, M., Arias, M., Doldan, S., Morsan, E.: Marine ecology service reuse through taxonomy-oriented SPL development. Comput. Geosci. **73**, 108–121 (2014). https://doi.org/10.1016/j.cageo.2014.09.004
7. Dhungana, D., GrÃnbacher, P., Rabiser, R.: The DOPLER meta-tool for decision-oriented variability modeling: a multiple case study. Autom. Softw. Eng. **18**(1), 77–114 (2011). https://doi.org/10.1007/s10515-010-0076-6
8. Hartmann, H., Trew, T.: Using feature diagrams with context variability to model multiple product lines for software supply chains. In: 2008 12th International Software Product Line Conference, pp. 12–21, September 2008. https://doi.org/10.1109/SPLC.2008.15
9. Kowal, M., Ananieva, S., Thüm, T.: Explaining anomalies in feature models. SIGPLAN Not. **52**(3), 132–143 (2016). https://doi.org/10.1145/3093335.2993248
10. Lisboa, L.B., Garcia, V.C., Lucrédio, D., Almeida, E.S., Meira, S.R.D.L., Fortes, R.P.M.: A systematic review of domain analysis tools. Inf. Softw. Technol. **52**(1), 1–13 (2010). https://doi.org/10.1016/j.infsof.2009.05.001
11. Mazo, R., Muñoz-Fernández, J., Rincón, L., Salinesi, C., Tamura, G.: VariaMos: an extensible tool for engineering (dynamic) product lines. In: Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, 20–24 July 2015, pp. 374–379 (2015). https://doi.org/10.1145/2791060.2791103
12. Mendonca, M., Branco, M., Cowan, D.: SPLOT: software product lines online tools. In: Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications, OOPSLA 2009, pp. 761–762. ACM, New York (2009). https://doi.org/10.1145/1639950.1640002
13. Metzger, A., Pohl, K., Heymans, P., Schobbens, P., Saval, G.: Disambiguating the documentation of variability in software product lines: a separation of concerns, formalization and automated analysis. In: 15th IEEE International Requirements Engineering Conference (RE 2007), pp. 243–253, October 2007. https://doi.org/10.1109/RE.2007.61
14. Pohl, K., Böckle, G., van Der Linden, F.J.: Software Product Line Engineering: Foundations. Principles and Techniques. Springer, New York (2005)
15. Pol'la, M., Buccella, A., Cechich, A.: Automated analysis of variability models: the SeVaTax process. In: Gervasi, O., et al. (eds.) ICCSA 2018. LNCS, vol. 10963, pp. 365–381. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-95171-3_29
16. Rincón, L., Giraldo, G., Mazo, R., Salinesi, C.: An ontological rule-based approach for analyzing dead and false optional features in feature models. Electron. Notes Theor. Comput. Sci. **302**, 111–132 (2014). https://doi.org/10.1016/j.entcs.2014.01.023
17. Roos-Frantz, F., Galindo, J., Benavides, D., Cortés, A.R.: FaMa-OVM: a tool for the automated analysis of OVMs. In: Proceedings of the 16th International Software Product Line Conference, vol. 2, pp. 250–254. ACM (2012)

18. Rosenmüller, M., Siegmund, N., Thüm, T., Saake, G.: Multi-dimensional variability modeling. In: Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems, VaMoS 2011, pp. 11–20. ACM, New York (2011). https://doi.org/10.1145/1944892.1944894
19. Segura, S., Benavides, D., Cortés, A.R.: Functional testing of feature model analysis tools. A first step. In: 12th International Conference Software Product Lines, Proceedings, Second Volume (Workshops), SPLC 2008, Limerick, Ireland, 8–12 September 2008, p. 179 (2008)
20. Sree-Kumar, A., Planas, E., Clariso, R.: Analysis of feature models using alloy: a survey. In: Proceedings 7th International Workshop on Formal Methods and Analysis in Software Product Line Engineering, FMSPLE@ETAPS 2016, Eindhoven, The Netherlands, 3 April 2016, pp. 46–60 (2016). https://doi.org/10.4204/EPTCS.206.5
21. Wohlin, C., Runeson, P., Hst, M., Ohlsson, M., Regnell, B., Wessln, A.: Experimentation in Software Engineering. Springer, Heidelberg (2012)
22. Zaid, L., Kleinermann, F., Troyer, O.D.: Applying semantic web technology to feature modeling. In: Proceedings of the 2009 ACM Symposium on Applied Computing, SAC 2009, pp. 1252–1256. ACM, New York (2009). https://doi.org/10.1145/1529282.1529563