An Automated Technique for Analysis of Orthogonal Variability Models based on Anti-patterns Detection using DL reasoning

Angela Oyarzun[1] and Germán Braun[1,3] and Laura Cecchi[1] and Pablo Fillottrani[2,4]

[1]Universidad Nacional del Comahue [2]Universidad Nacional del Sur
[3]*Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET)*
[4]*Comisión de Investigaciones Científicas de la Provincia de Buenos Aires (CIC)*
angela.oyarzun@est.fi.uncoma.edu.ar
{german.braun,lcecchi}@fi.uncoma.edu.ar
prf@cs.uns.edu.ar

Abstract During a Software Product Line (SPL) variability management, model validation is crucial so as to detect faults in early development stages and avoid affecting derived products quality. Therefore, the automated variability analysis has emerged for translating and validating variability models. In this work, we present a catalogue of anti-patterns, which describes scenarios associated to the detection of problems in a SPL. Moreover, we extend *crowd-variability*, a novel graphical tool designed for modelling and validating Orthogonal Variability Models (OVM), for detecting such anti-patterns using Description Logics (DL)-based reasoning services.

Keywords: Software Product Lines, Orthogonal Variability Models, Description Logics, Graphical tools for modelling variability

1 Introduction

Currently, the increasing complexity of Information Systems, derived from a Software Product Line (SPL), results in more complex variability models (VM). Consequently, the tasks involved in variability management become more costly in time and error prone, making it impossible to guarantee the quality of final products without the assistance of automated tools. In this regard, we developed *crowd-variability*[1], a graphical tool for designing, visualising and validating orthogonal variability models (OVM). This novel client-server tool, presented in [1], provides graphical support for users modelling their diagrams and is integrated with automatic DL-based reasoning.

In variability management of SPLs, checking consistency of variability models (VM) is a critical problem. On this line, the most common mistake committed when modelling variability of a SPL is invalidating its services, i.e. creating dead services. This problem directly affects the quality of products as it can invalidate a whole model. Therefore, we need a way of capturing this error-prone modelling decision. For this purpose, there are anti-patterns [2].

[1] `http://crowd.fi.uncoma.edu.ar/crowd-variability/web-src/`

In this work, we extend *crowd-variability* automated variability analysis support by detecting a series of anti-patterns which are based on a set of instances of dead services. Each of these anti-patterns corresponds with one or more dependencies of variability and/or restrictions of a model. Furthermore, we simplified its original DL $\mathcal{ALCI}$ encoding and we extended it in order to be able to classify services and dependencies according to its type. A prototype of this tool already runs on a client-server architecture and enables anti-patterns detection on OVM graphical diagrams.

This work is structured as follows. Section 2 introduces orthogonal variability models, describes automated variability analysis and compares *crowd-variability* with other existing tools. Section 3 presents anti-patterns and their detection algorithms. Section 4 presents the prototype developed together with a simple example of use and exposes a preliminary evaluation and discussions. To conclude the paper, section 5 elaborates on final considerations and directions for future works.

2 Context

Orthogonal Variability Models (OVM) A software product line is a means to develop a set of products in which variability is a central phenomenon captured in variability models [3]. Among these models, there are Orthogonal Variability Models comprised by services, variants and variation points, related by variability and constraint dependencies. Both elements and interactions are depicted in Table 1 using *crowd-variability*'s graphical language which is based on the one presented in [4]. We expanded JointJS library functionality by developing a plugin for these OVM primitives.

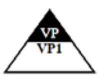
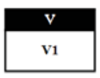
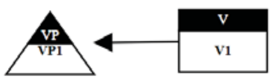
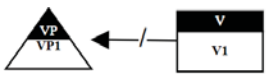
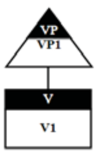
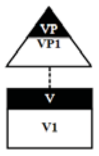
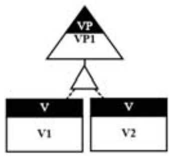
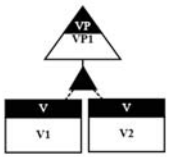
Variation Point	Variant	Requires	Excludes
			
Mandatory	Optional	Alternative	Variant
			

Table 1. OVM Graphical Components

Services
- *Variation Point*: It represents a variable object in real world.
- *Variant*: It represents different forms in which a variation point varies.

Constraint Dependencies: Interactions between variation points, variants or a variant and a variation point.
- *Requires*: It describes the selection of a service requires the selection of its associated service.

- *Excludes*: It describes the selection of a service excludes the selection of its associated service.

Variability Dependencies: Interactions between a variation point and a group of variants and/or variation points.
- *Mandatory*: It describes the selection of a variation point requires the selection of all associated services.
- *Optional*: It describes the selection of a variation point requires the selection of 0 or more associated services.
- *Alternative*: It describes the selection of a variation point requires the selection of only one of the associated services.
- *Variant*: It describes the selection of a variation point requires the selection of at least one of the associated services.

Automated Variability Analysis Variability Management is one of the most important activities in a SPL as it has a great impact on the way software is developed, extended and maintained. Therefore, automated variability analysis has emerged [5, 6, 7, 8], which bases on the extraction of information from models using automated mechanisms. During this process, a variability model is translated into a formal representation, that is sent to a reasoner in order to validate such model, taking into consideration the anomalies and incompatibilities it may contain. The results of this process answer some of the following queries:
- **Valid Model** (VM): Given a model, it determines whether at least one product can be derived from it.
- **Valid Instantiation** (VI): Given a model and one of its possible instantiations, it determines whether such instantiation is a valid product.
- **All Products** (AP): Given an OVM, it returns all possible instantiations.
- **Instantiation Number** (IN): Given an OVM, it indicates the number of possible instantiations.
- **Problem Detection** (PD): It identifies the causes or anomalies that invalidate the model and returns this information.
- **Cross Validation** (CV): It evaluates the validity between models that use the same service.

In particular, during the detection of problems, this undesirable information [6] about models should be identified:
- **Dead Service**: It is a service that will never be derived as part of a product [9].
- **False Optional Service**: Given the selection of its parents, it is a service that is included in all derived products of a product line in spite of not being modelled as mandatory.
- **Wrong Cardinality**: A group or range of cardinality is wrong if it can never be instantiated.
- **Redundancy**: Occurs when semantic information is modelled in different ways.

Comparison with other tools We have surveyed existing tools for automated variability analysis taking into consideration graphical support, variability model, formalisation method and automatic reasoning support. FaMa-OVM [10] is an extensible tool for automated analysis of OVM diagrams, integrated with multiple off.the-shelf reasoners such as SAT4j, JavaBDD and Choco-Solver. However its input model is specified in a textual format, which makes the model specification a difficult and error-prone task as users have to learn this new syntax. Furthermore, due to the fact that it works with SAT reasoners, it presents certain limitations related to more restrictive logics and thus fail to reflect the finer logical structure of variability models. On the other hand, VariaMos [11] is a tool that allows defining modelling languages and automatically analysing the generated models. This tools consists of a graphical "front-end" and a "back-end" that implements all required functionalities. It is independent from the variability modelling language and works with two reasoners, SWI Prolog and GNU Prolog. Nonetheless, it does not use the standard OVM graphical language, forcing users to learn this new language in order to properly design their models.

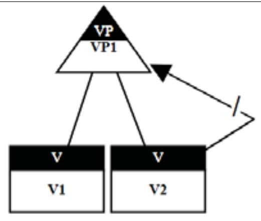
In contrast with these tools, *crowd-variability* is a tool that provides users with a graphical interface with all the necessary functionalities for representing OVM diagrams, using the standard graphical language, and is integrated with DL-based reasoning systems which are in charge of carrying out a more precise variability analysis.

3 An Automated Technique for OVM Analysis

According to the previously described problems, we focus on *dead services* detection for presenting a technique for automated analysis of OVM based on anti-patterns. We emphasise on dead services because they are the most influential in the quality of derived products of a SPL [12]. A service is dead if it cannot appear in any of the generated products of the software product line. Additionally, we also consider the validation scenario named *cross validation* as another modelling failure, where a variant belongs to two or more variation points in the same model. So in order to find these described problems in the context of more complex OVM models, a set of anti-patterns capturing modelling scenarios has been proposed: Mandatory Exclude, Parent Exclude, Alternative Ambiguity, Transitivity Contradiction, Constraint Contradiction, and Cross Validation. These instances were extracted from [12] and renamed for this work. For detecting them, we have extended DL $\mathcal{ALCI}$ encoding proposed in [9], so as to be able to query the type of the concepts and roles defined in an OVM. According to the encoding in [9], each service is encoded by a concept and each relation between them in a constraint or a variability dependency is encoded by a role. The extended encoding includes new concepts and axioms, so as to classify services and dependencies. Furthermore, we have designed and implemented a series of algorithms that have as input the reasoner answers for queries on services and their relationships. For instance, for the mandatory variability dependency as indicated in Table 1, our proposal identifies such dependency modelled as a DL role and after that it gets both domain (VP1) and range (V1) of such mandatory role.

Now we present our catalogue of anti-patterns, explain and illustrate each one of them, and provide a DL encoding and an algorithm for detecting them in an OVM model. Such anti-patterns detection algorithms are tractable. The time complexity is nm for each one, where n is the cardinality of object properties and m is the number of relations in the model.

Mandatory Exclude A service excludes a mandatory service. In this scenario, a derived product from this model must include the services VP1, V1 and V2, however, VP1 and V2 are mutually excluded.



$VP1$ MAND$\{V1, V2\}$

$\exists vp1v1 \sqsubseteq VP1$ $\exists vp1v2 \sqsubseteq VP1$ (Domain)

$\exists vp1v1^{-} \sqsubseteq V1$ $\exists vp1v2^{-} \sqsubseteq V2$ (Range)

$vp1v1 \sqsubseteq MandatoryDep$ $vp1v2 \sqsubseteq MandatoryDep$

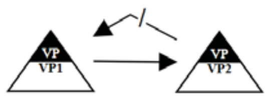
If $V2$ EXC $VP1$, then:

$\exists E_{v2vp1} \sqsubseteq V2$ (Domain) $\exists E^{-}_{v2vp1} \sqsubseteq VP1$ (Range)

$v2vp1 \sqsubseteq ExcludesDep$

How to detect it? Both domain and range of the excludes dependency should belong to the set of domain and range of the mandatory dependency. This is done by Algorithm 1, where the set of object properties (*ObjectPropertySynset*) for mandatory and excludes are revised and compared.

Transitivity Contradiction A service includes another one that excludes the first one. In this scenario, a derived product from this model applying the *requires* dependency must include the services VP1 and VP2, however, VP2 excludes VP1 according to the respective dependency.



If $VP1$ REQ $VP2$, then:

$\exists R_{vp1vp2} \sqsubseteq VP1$ (Domain) $\exists R^{-}_{vp1vp2} \sqsubseteq VP2$ (Range)

$vp1vp2 \sqsubseteq RequiresDep$

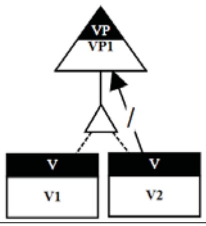
If $VP2$ EXC $VP1$, then:

$\exists E_{vp2vp1} \sqsubseteq VP2$ (Domain) $\exists E^{-}_{vp2vp1} \sqsubseteq VP1$ (Range)

$vp2vp1 \sqsubseteq ExcludesDep$

How to detect it? The domain and range of the excludes dependency should be equal to the range and domain of the requires dependency, respectively. This is done by Algorithm 2, where the set of object properties (*ObjectPropertySynset*) for requires and excludes are revised and compared.

Parent Exclude One or more alternative, variant, or optional services exclude its parent. In this scenario, a derived product from this model must include VP1 and only one of the services V1 and V2, however, V2 excludes VP1.



$VP1$ ALT$\{V1, V2\}$

$\exists vp1v1 \sqsubseteq VP1$ $\exists vp1v2 \sqsubseteq VP1$ (Domain)

$\exists vp1v1^{-} \sqsubseteq V1$ $\exists vp1v2^{-} \sqsubseteq V2$ (Range)

$vp1v1 \sqsubseteq AlternativeDep$ $vp1v2 \sqsubseteq AlternativeDep$

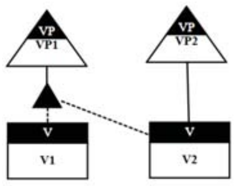
If $V2$ EXC $VP1$, then:

$\exists E_{v2vp1} \sqsubseteq V2$ (Domain) $\exists E^{-}_{v2vp1} \sqsubseteq VP1$ (Range)

$v2vp1 \sqsubseteq ExcludesDep$

How to detect it? Both domain and range of the excludes dependency should belong to the set of domain and range of the alternative (or optional or variant) dependency. This is done by Algorithm 3, where the set of object properties (*ObjectPropertySynset*) for alternative and excludes are revised and compared.

Cross Validation A same service belongs to two or more variation points. In this scenario, a derived product from this model must include the services VP2, V2, VP1, however, V1 may or not be included.



$VP1$ VAR$\{V1, V2\}$

$\exists vp1v1 \sqsubseteq VP1$ $\exists vp1v2 \sqsubseteq VP1$ (Domain)

$\exists vp1v1^- \sqsubseteq V1$ $\exists vp1v2^- \sqsubseteq V2$ (Range)

$vp1v1 \sqsubseteq VariantDep$ $vp1v2 \sqsubseteq VariantDep$

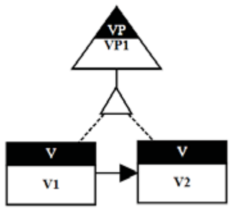
$VP2$ MAND$\{V2\}$

$\exists vp2v2 \sqsubseteq VP2$ (Domain) $\exists vp2v2^- \sqsubseteq V2$ (Range)

$vp2v2 \sqsubseteq MandatoryDep$

How to detect it? The range of the variability dependencies should match in at least one element. This is done by Algorithm 4, where the set of object properties (*ObjectPropertySynset*) for variant and mandatory are revised and compared.

Alternative Ambiguity A service includes an alternative of itself or an alternative of its parents. In this scenario, a derived product from this model must include the services VP1 and only one of the services V1 and V2, however, V1 requires V2 to be included.



$VP1$ ALT$\{V1, V2\}$

$\exists vp1v1 \sqsubseteq VP1$ $\exists vp1v2 \sqsubseteq VP1$ (Domain)

$\exists vp1v1^- \sqsubseteq V1$ $\exists vp1v2^- \sqsubseteq V2$ (Range)

$vp1v1 \sqsubseteq AlternativeDep$ $vp1v2 \sqsubseteq AlternativeDep$

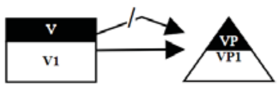
If $V1$ REQ $V2$, then:

$\exists R_{v1v2} \sqsubseteq V1$ (Domain) $\exists R^-_{v1v2} \sqsubseteq V2$ (Range)

$v1v2 \sqsubseteq RequiresDep$

How to detect it? Both domain and range of the requires dependency should belong to the set of range of the alternative dependency. This is done by Algorithm 5, where the set of object properties (*ObjectPropertySynset*) for alternative and requires are revised and compared.

Constraint Contradiction A service includes and excludes the same service, presents a transitive relationship of this case, or presents a dependency with itself. In this scenario, a derived product from this model must include the services V1 and VP1, however, V1 excludes VP1.



If $V1$ REQ $VP1$, then:

$\exists R_{v1vp1} \sqsubseteq V1$ (Domain) $\exists R^-_{v1vp1} \sqsubseteq VP1$ (Range)

$v1vp1 \sqsubseteq RequiresDep$

If $V1$ EXC $VP1$, then:

$\exists E_{v1vp1} \sqsubseteq V1$ (Domain) $\exists E^-_{v1vp1} \sqsubseteq VP1$ (Range)

$v1vp1 \sqsubseteq ExcludesDep$

How to detect it? The domain and range of the restriction dependencies should be equal. This is done by Algorithm 6, where the set of object properties (*ObjectPropertySynset*) for requires and excludes are revised and compared.

Algorithm 1: Mandatory Exclude

```
1: AntiPattern = []
2: Mandatories = []
3: for all ObjectPropertySynset mand in GetSubObjectProperties(MandatoryDep) do
4:     Add Domain(mand) $\rightarrow$ Range(mand) to Mandatories
5: end for
6: for all ObjectPropertySynset exc in GetSubObjectProperties(ExcludesDep) do
7:     for all origin $\rightarrow$ targets in Mandatories do
8:         if $\{$Domain(exc), Range(exc)$\}$ is included in $\{$origin, targets$\}$ then
9:             Add $\{$origin, targets$\}$ to AntiPattern
10:        end if
11:    end for
12: end for
13: return AntiPattern
```

Algorithm 2: Trans. Contradiction

```
1: AntiPattern = []
2: for all ObjectPropertySynset exc in GetSubObjectProperties(ExcludesDep) do
3:     for all ObjectPropertySynset req in GetSubObjectProperties(RequiresDep) do
4:         if Domain(exc) = Range(req) and Domain(req) = Range(exc) then
5:             Add $\{$Domain(exc), Range(exc)$\}$ to AntiPattern
6:         end if
7:     end for
8: end for
9: return AntiPattern
```

Algorithm 3: Parent Exclude

```
1: AntiPattern = []
2: Optionals = []
3: Alternatives = []
4: Variants = []
5: for all ObjectPropertySynset opt in GetSubObjectProperties(OptionalDep) do
6:     Add Domain(opt) $\rightarrow$ Range(opt) to Optionals
7: end for
8: for all ObjectPropertySynset alt in GetSubObjectProperties(AlternativeDep) do
9:     Add Domain(alt) $\rightarrow$ Range(alt) to Alternatives
10: end for
11: for all ObjectPropertySynset var in GetSubObjectProperties(VariantDep) do
12:     Add Domain(var) $\rightarrow$ Range(var) to Variants
13: end for
14: for all ObjectPropertySynset exc in GetSubObjectProperties(ExcludesDep) do
15:     for all origin $\rightarrow$ targets in Optionals do
16:         if Range(exc) = origin and $\{$Domain(exc)$\}$ is included in targets then
17:             Add $\{$origin, targets$\}$ to AntiPattern
18:         end if
19:     end for
20:     for all origin $\rightarrow$ targets in Alternatives do
21:         if Range(exc) = origin and $\{$Domain(exc)$\}$ is included in targets then
22:             Add $\{$origin, targets$\}$ to AntiPattern
23:         end if
24:     end for
25:     for all origin $\rightarrow$ targets in Variants do
26:         if Range(exc) = origin and $\{$Domain(exc)$\}$ is included in targets then
27:             Add $\{$origin, targets$\}$ to AntiPattern
28:         end if
29:     end for
30: end for
31: return AntiPattern
```

Algorithm 4: Cross Validation

```
1: AntiPattern = []
2: Targets=[]
3: for all ObjectPropertySynset mand in GetSubObjectProperties(MandatoryDep) do
4:     Add Range(mand) to Targets
5: end for
6: for all ObjectPropertySynset opt in GetSubObjectProperties(OptionalDep) do
7:     Add Range(opt) to Targets
8: end for
9: for all ObjectPropertySynset alt in GetSubObjectProperties(AlternativeDep) do
10:     Add Range(alt) to Targets
11: end for
12: for all ObjectPropertySynset var in GetSubObjectProperties(VariantDep) do
13:     Add Range(var) to Targets
14: end for
15: Add getDuplicates(Targets) to AntiPattern
16: return AntiPattern
```

Algorithm 5: Alt. Ambiguity

```
1:  AntiPattern = []
2:  Alternatives = []
3:  for all ObjectPropertySynset alt in GetSubObjectProperties(AlternativeDep) do
4:      Add Domain(alt) → Range(alt) to Alternatives
5:  end for
6:  for all ObjectPropertySynset req in GetSubObjectProperties(RequiresDep) do
7:      for all origin → targets in Alternatives do
8:          if {Domain(req), Range(req)} is included in targets then
9:              Add {origin, targets} to AntiPattern
10:         end if
11:     end for
12: end for
13: return AntiPattern
```

Algorithm 6: Cons. Contradiction

```
1:  AntiPattern = []
2:  for all ObjectPropertySynset exc in GetSubObjectProperties(ExcludesDep) do
3:      if (Domain(exc) = Range(exc)) then
4:          Add Domain(exc) to AntiPattern
5:      end if
6:      for all ObjectPropertySynset req in GetSubObjectProperties(RequiresDep) do
7:          if ( (Domain(exc) = Domain(req) and Range(req) = Range(exc)) or (Domain(req) = Range(req)) ) then
8:              Add {Domain(req), Range(req)} to AntiPattern
9:          end if
10:     end for
11: end for
12: return AntiPattern
```

4 *crowd-variability* for Anti-patterns Detection

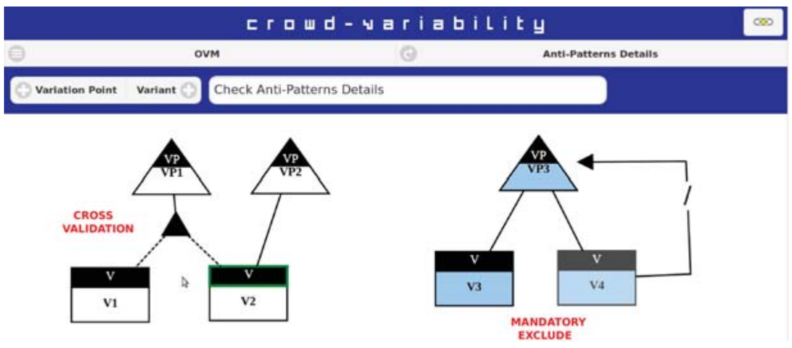


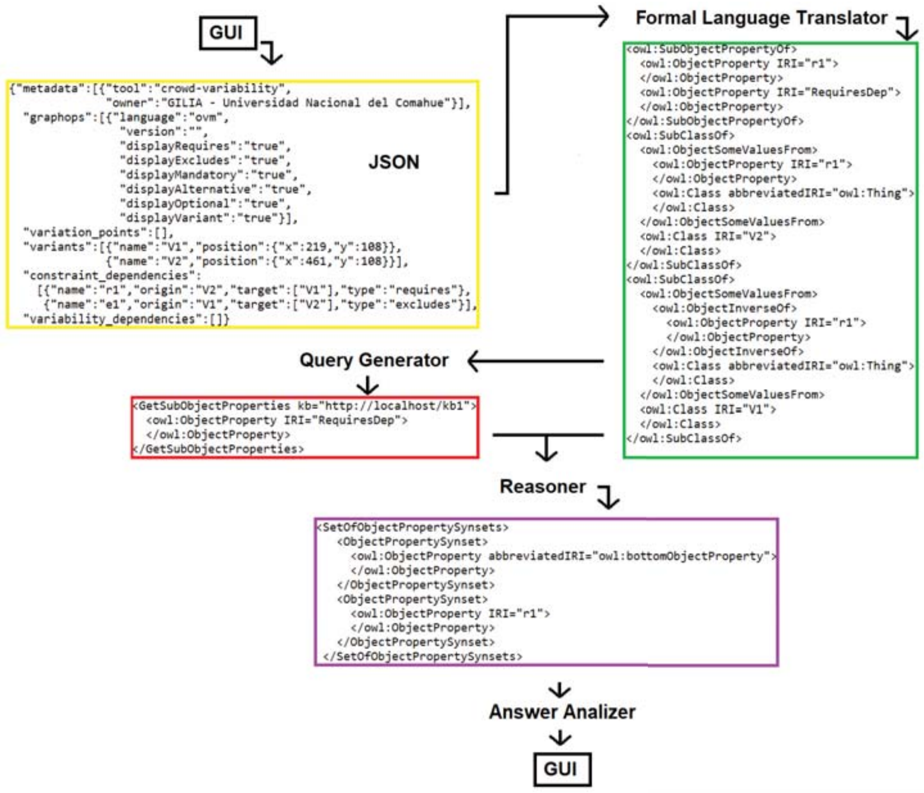
Figure 1. *crowd-variability* Front-End

New prototype of *crowd-variability* is integrated with Racer [13] and SPARQL-DL [14] reasoners. For this reason, it offers users the possibility of requesting an anti-pattern detection of a model by pressing the semaphore presented in Figure 1. By doing so, a JSON representation of the model is sent to the server, through a HTTP request, in order to be formalized in $\mathcal{ALCI}$ DL by the Formal Language Translator. This module generates a document in OWL syntax and another one in OWLlink syntax with the formal representation of the model. Later, The Query Generator creates a series of queries for each document. The OWLlink document, comprised of the model's formalisation and its queries, is sent to RACER reasoner and the OWL document with its queries are sent separately to SPARQL-DL reasoner. The results generated by the reasoners are later processed by the Answer Analyser. This module produces a JSON file with information about anti-patterns occurrences in the diagram. Finally, the client side receives this JSON file and graphically presents such information to the user by colouring those services involved in anti-patterns. Each anti-pattern has a corresponding colour associated. Figure 2 presents the path followed by a model starting in the

GUI for its creation and ending in the GUI with all its services involved in anti-pattern coloured and listed (see Figure 1). Other tests can be found by following this link: `https://bitbucket.org/gilia/crowd-variability/wiki/`.

Algorithms for anti-patterns detection were designed for recognising the scenarios presented in the catalogue. Therefore, their recognition is made at only one level of depth of the models. However, they can be generalised in order to detect dead services in larger diagrams. Likewise, new anti-patterns can be added to the catalogue and their associated algorithms for detection can be implemented in *crowd-variability*.

Lastly, *crowd-variability* has been developed using expansible graphical libraries and Web technologies. Therefore, it can be extended to other variability modelling proposals by creating or adding new plug-ins to the JointJs graphics library, incorporating new formalisation methods, or connecting new back-end reasoners. Hence, a relevant feature of crowd variability is to consider users' preferences and usages, allowing the selection of different approaches to model variability and distinct reasoners for the reasoning service.



```
{"metadata":[{"tool":"crowd-variability",
              "owner":"GILIA - Universidad Nacional del Comahue"}],
  "graphops":[{"language":"ovm",
               "version":"",
               "displayRequires":"true",
               "displayExcludes":"true",
               "displayMandatory":"true",
               "displayAlternative":"true",
               "displayOptional":"true",
               "displayVariant":"true"}],
  "variation_points":[],
  "variants":[{"name":"V1","position":{"x":219,"y":108}},
              {"name":"V2","position":{"x":461,"y":108}}],
  "constraint_dependencies":
    [{"name":"r1","origin":"V2","target":["V1"],"type":"requires"},
     {"name":"e1","origin":"V1","target":["V2"],"type":"excludes"}],
  "variability_dependencies":[]}
```

```
<owl:SubObjectPropertyOf>
  <owl:ObjectProperty IRI="r1">
  </owl:ObjectProperty>
  <owl:ObjectProperty IRI="RequiresDep">
  </owl:ObjectProperty>
</owl:SubObjectPropertyOf>
<owl:SubClassOf>
  <owl:ObjectSomeValuesFrom>
    <owl:ObjectProperty IRI="r1">
    </owl:ObjectProperty>
    <owl:Class abbreviatedIRI="owl:Thing">
    </owl:Class>
  </owl:ObjectSomeValuesFrom>
  <owl:Class IRI="V2">
  </owl:Class>
</owl:SubClassOf>
<owl:SubClassOf>
  <owl:ObjectSomeValuesFrom>
    <owl:ObjectInverseOf>
      <owl:ObjectProperty IRI="r1">
      </owl:ObjectProperty>
    </owl:ObjectInverseOf>
    <owl:Class abbreviatedIRI="owl:Thing">
    </owl:Class>
  </owl:ObjectSomeValuesFrom>
  <owl:Class IRI="V1">
  </owl:Class>
</owl:SubClassOf>
```

```
<GetSubObjectProperties kb="http://localhost/kb1">
  <owl:ObjectProperty IRI="RequiresDep">
  </owl:ObjectProperty>
</GetSubObjectProperties>
```

```
<SetOfObjectPropertySynsets>
   <ObjectPropertySynset>
     <owl:ObjectProperty abbreviatedIRI="owl:bottomObjectProperty">
     </owl:ObjectProperty>
   </ObjectPropertySynset>
   <ObjectPropertySynset>
     <owl:ObjectProperty IRI="r1">
     </owl:ObjectProperty>
   </ObjectPropertySynset>
</SetOfObjectPropertySynsets>
```

Figure 2. *crowd-variability* Back-End

5 Conclusions and Future Work

In this work, we presented a catalogue of anti-patterns for automated analysis of variability models. Such anti-patterns match specific modelling scenarios obtained from the literature. Moreover, we extend our tool *crowd-variability* for detecting them by extending the DL encoding of OVM and providing tractable algorithms for processing reasoning outputs. In spite of the fact that this is a starting catalogue, our technique detects dead services, which are the most influential in the quality of derived products of a SPL. However, the approach presents some limitations associated to the complexity of models because the anti-patterns detection is made at only one level of depth in the models.

In future works, we plan to extend its support for automated variability analysis by generalising anti-patterns detection algorithms, expanding the catalogue and considering other queries about the model. Furthermore, we will continue evaluating *crowd-variability* so as to enhance and extend its functionalities.

References

1. A. Oyarzun, G. Braun, L. Cecchi, and P. Fillottrani. A graphical web tool with dl-based reasoning support over orthogonal variability models. In *XXIV CACIC.*, 2018.
2. T. Sales and G. Guizzardi. Ontological anti-patterns: Empirically uncovered error-prone structures in ontology-driven conceptual models. *Data & Knowledge Engineering*, 2015.
3. M. Raatikainen, J. Tiihonen, and T. Männistö. Software product lines and variability modeling: A tertiary study. *Journal of Systems and Software*, 2019.
4. K. Pohl, G. Böckle, and F. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. 2005.
5. F. Roos-Frantz, J. Galindo, D. Benavides, A. Ruiz-Cortés, and J. Garcıa-Galán. Automated analysis of diverse variability models with tool support. *Jornadas de Ingenierıa del Software y de Bases de Datos (JISBD 2014)*, 2014.
6. D. Benavides, S. Segura, and A. Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. *Inf. Syst.*, 2010.
7. M. Kowal, S. Ananieva, and T. Thüm. Explaining anomalies in feature models. In *ACM SIGPLAN*, GPCE 2016. ACM, 2016.
8. A. Metzger, K. Pohl, P. Heymans, P. Y. Schobbens, and G. Saval. Disambiguating the Documentation of Variability in Software Product Lines: A Separation of Concerns, Formalization and Automated Analysis. In *15th IEEE IREC*, 2007.
9. G. Braun, M. Pol'la, L. Cecchi, A. Buccella, P. Fillottrani, and A. Cechich. A DL Semantics for Reasoning over OVM-based Variability Models. In *Description Logics*, 2017.
10. F. Roos Frantz, J. Galindo Duarte, D. Benavides Cuevas, and A. Ruiz Cortés. FaMa-OVM: A tool for the automated analysis of ovms. In *16th ISPLC*, 2012.
11. R. Mazo, J. Muñoz-Fernández, L. Rincón-Perez, C. Salinesi, and G. Tamura. VariaMos: an extensible tool for engineering (dynamic) product lines. In *SPLC 2015*, 2015.
12. M. Pol'la, A. Buccella, and A. Cechich. Automated analysis of variability models: The SeVaTax Process. In *ICCSA*. Springer, 2018.
13. V. Haarslev and R. Möller. RACER System Description. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Automated Reasoning*, pages 701–705, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.
14. E. Sirin and B. Parsia. SPARQL-DL: SPARQL Query for OWL-DL. In *In 3rd OWL Experiences and Directions Workshop (OWLED-2007)*, 2007.