Delft University of Technology
Software Engineering Research Group
Technical Report Series

Evaluating Feature Change Impact on Multi-Product Line Configurations Using Partial Information

Nicolas Dintzner, Uira Kulesza, Arie van Deursen and Martin Pinzger

Report TUD-SERG-2014-018

TUD-SERG-2014-018

Published, produced and distributed by:

Software Engineering Research Group
Department of Software Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4
2628 CD Delft
The Netherlands

ISSN 1872-5392

Software Engineering Research Group Technical Reports:
http://www.se.ewi.tudelft.nl/techreports/

For more information about the Software Engineering Research Group:
http://www.se.ewi.tudelft.nl/

Note: Accepted for publication in the Proceedings of 14th International Conference on Software Reuse (ICSR 2015)

Evaluating Feature Change Impact on Multi-Product Line Configurations Using Partial Information

Nicolas Dintzner[1], Uirá Kulesza[2], Arie van Deursen[1], and Martin Pinzger[3]

[1] Software Engineering Research Group, Delft University of Technology, Delft, Netherlands,
{N.J.R.Dintzner},{Arie.vanDeursen}@tudelft.nl
[2] Department of Informatics and Applied Mathematics, Federal University of Rio Grande do Norte, Natal, Brazil,
uira@dimap.ufrn.br
[3] Software Engineering Research Group, University of Klagenfurt, Klagenfurt, Austria,
martin.pinzger@aau.at

Abstract. Evolving large-scale, complex and highly variable systems is known to be a difficult task, where a single change can ripple through various parts of the system with potentially undesirable effects. In the case of product lines, and moreover multi-product lines, a change may affect only certain variants or certain combinations of features, making the evaluation of change effects more difficult.
In this paper, we present an approach for computing the impact of a feature change on the existing configurations of a multi-product line, using partial information regarding constraints between feature models. Our approach identifies the configurations that can no longer be derived in each individual feature model taking into account feature change impact propagation across feature models. We demonstrate our approach using an industrial problem and show that correct results can be obtained even with partial information. We also provide the tool we built for this purpose.

Keywords: product line, variability, change impact, feature, model-based

1 Introduction

Evolving large-scale, complex and variable systems is known to be a difficult task, where a single change can ripple through various parts of the system with potentially undesirable effects. If the components of this system are themselves variable, or if the capabilities exposed by an interface depend on some external constraint (i.e. configuration option), then engineers need extensive domain knowledge on configuration options and component implementations to safely improve their system [8]. In the domain of product line engineering (PLE), an

approach aiming at maximising asset reuse in different products [14], this type of evolutionary challenge is the norm. Researchers and practitioners have looked into what variability modeling - and feature modeling specifically - can bring to change impact analysis on product lines (PLs). Existing methods can evaluate, given a change expressed in features, how a feature model (FM) and the composition of features it allows (configurations) are impacted [7, 13, 19]. However, FMs grow over time in terms of number of features and constraints and safe manual updates become unmanageable by humans [4]. Moreover, automated analysis methods do not scale well when the number of configurations or feature increases [7].

To mitigate this, *nested product lines*, *product populations*, or *multi-product lines* (MPL - a set of interdependent PLs) approaches recommend modularizing FMs into smaller and more manageable pieces [11, 12, 18]. While this solves part of the problem, known FM analysis methods are designed for single FMs. A common approach is to recompose the FMs into a single one. To achieve this, existing approaches suggest describing explicitely dependencies between FMs using cross-FM constraints, or hierarchies [1] to facilitate model composition and analysis. Such relationships act as vectors of potential change impact propagation between FMs. However, in [9] Holl et al. noted that the knowledge of domain experts about model constraints is likely to be only partial (both intra-FMs or extra-FMs). For this reason, we cannot assume that such relationships will be available as inputs to a change impact analysis.

In this context, we present and evaluate an approach to facilitate the assessment of the impact of a feature change on existing configurations of the different PLs of an MPL using partial information about inter-FMs relationships. After giving background information regarding feature modeling and product lines (Section 2), we present the industrial problem that motivated this work and detail the goals and constraints of this study (Section 3). We then present our approach to enrich the variability model of an MPL using existing configurations of individual FMs, and the heuristic we apply when analyzing the effect of a feature change on existing configurations of an MPL (Section 4). In Section 5, we assess our approach in an industrial context. We present and discuss how we built the appropriate models, the output of our prototype implementation and the performance of the approach with its limitations. Finally, Section 6 presents related work and we elaborate on possible future work in Section 7.

2 Background

In this paper, the definition of *feature* given by Czarnecki et al. in [5] is used: "a feature may denote any functional or nonfunctional characteristic at the requirement, architectural, component, platform or any other level". A feature model (FM) is a structured set of features with selection rules specifying the allowed combinations of features. This is achieved through relationships (optional, mandatory, part of an alternative or OR-type structures) and cross-tree constraints - arbitrary conditions on feature selection. The most common types of

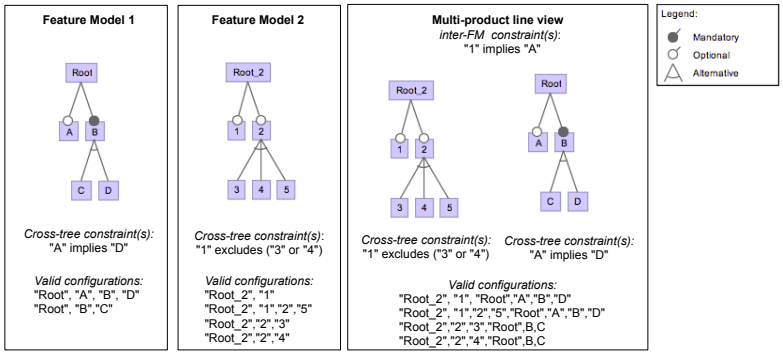

Fig. 1. Example of FMs in a SPL and MPL context

cross-tree constraints are "excludes" (e.g. "feature A excludes feature B") and "implies" [10]. With a FM, one can derive configurations: a set of features which does not violate constraints established by the FM. An example of simple FMs with their valid configurations are depicted on the left hand side of Figure 1.

In the context of a multi-product line, several inter-related FMs are used to describe the variability of a single large system. This can be achieved by creating "cross-feature model" constraints or through feature references [3] - where a given feature appears in multiple FMs. The constraints between FMs can be combination rules referring to features contained within different models. Those constraints can also be derived from the hierarchy (or any imposed structure [3, 15]) of the FMs involved in an MPL. In those cases, the combination rules can refer to both features and FMs. A product configuration derived from an MPL is a set of features which does not violate any constraints of individual FMs nor the cross-FM constraints that have been put in place. An example of combined FMs with a constraint between two FMs can be seen on the right hand side of Figure 1.

3 Motivation: change impact in an industrial context

Our industrial partner builds and maintains high-end medical devices, among which an x-ray machine. This x-ray machine comes in many variants, each differing in terms of hardware (e.g. tables, mechanical arms) and software (e.g. firmware version, imaging system). Certified third party products can be integrated through different types of external interfaces: mechanical (e.g. a module placed on the operating table), electrical (inbuilt power supply), data related (image transfer). As an example, three main subsystems of the x-ray machine (data/video exchange, video chain, and display) and three main interfaces (display interface, video signal, and data/video exchange) are shown in Figure 2. The two working modes of a given 3rd party product ("mode 1" and "mode 2") use the same interfaces in slightly different ways. In "mode 1", the 3rd party product reuses the x-ray machine display to show images ("shared display") while in

4

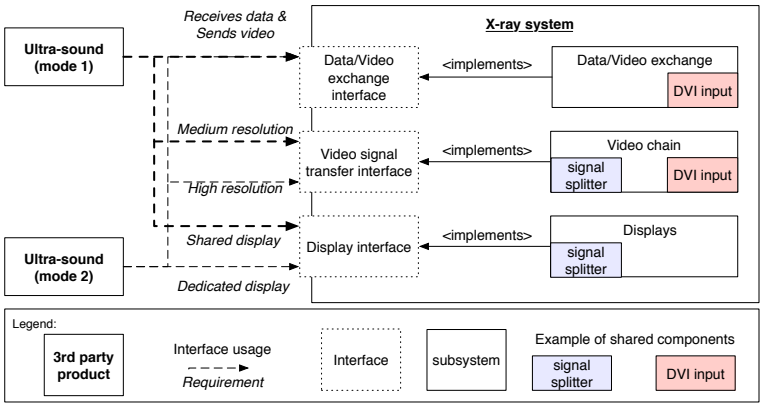

Fig. 2. X-ray machine system overview

"mode 2" a dedicated display is used. Sharing an existing display implies using a signal splitter/merger in the display subsystem. But the splitter/merger also plays a role in the video processing chain and is only available in certain of its variants.

Following any update, engineers must validate if the new version of the system can still provide what is necessary for 3rd party product integration. This leads to the following type of questions: *"Knowing that 3rd party product X uses the video interface to export high resolution pictures and import patient data, is X supported by the new version of the x-ray machine?"*. Let us consider the following scenario: a connection box, present in the video chain and data/video exchange subsystems, is removed from the list of available hardware. Some specific configurations of the video chain and of the data/video exchange subsystems can no longer be produced. The data/video exchange interface required the removed configurations to provide specific capabilities. Following this, it is no longer possible to export video and import data and the integration with the 3rd party product is compromised.

Currently, engineers validate changes manually by checking specification documents (either 3rd party products requirements or subsystem technical specifications) and rigorous testing practices. Despite this, it remains difficult to assess which subsystem(s) and which of their variant(s) or composition of variants will be influenced by a given change. Given the rapid evolution of their products, this error-prone validation is increasingly time consuming. Our partner is exploring model-driven approaches enabling early detection of such errors.

While this example is focused on the problems that our industrial partner is facing, enabling analysis for very large PLs and MPLs is a key issue for many companies. Recently, Schmid introduced the notion of variability-rich eco systems [17], highlighting the many sources of variability that may influence a software product. This further emphasizes the need for change impact analysis approaches on highly variable systems.

4 Feature-change impact computation

Given the problem described in the previous section, we present here the approach we designed to assist domain engineers in evaluating the impact of a change on their products. We first describe the main goal of our approach and our contributions. Then, we detail the approach and illustrate it with a simple example. Finally, we consider the scalability aspects of the approach and present our prototype implementation.

4.1 Goals and constraints

For our industrial partner, the main aim is to obtain insights on the potential impacts of an update on external interfaces used by 3rd party products. However, we have to take into account that domain engineers do not know the details of the interactions of the major subsystems [9] nor all components included in each one - only the ones relevant to support external interfaces. As an input, we rely on the specifications of each major subsystem and their main components in isolation as well as their existing configurations. Because of the large number of subsystem variants and interface usages (choices of capabilities or options), we consider each of them as a product line (PL) in its own right. Features then represent hardware components, (non-)functional properties, software elements, or any other relevant characteristic of a subsystem or interface. Using a simple feature notation and cross-tree constraints [10], we formalize within each subsystem the known features and constraints between them. By combining those PLs, we obtain a multi-product line (MPL) representation of the variability of the system.

With such representation, a change to a subsystem or interface can be expressed in terms of features: adding or removing features, adding, removing or modifying intra-FM constraints. Once the change is known, we can apply it to the relevant FM and evaluate if existing configurations are affected (no longer valid with respect to the FM). Then, we determine how the change propagates across the FMs of the MPL using a simple heuristic on configuration composition. As an ouput, we provide a tree of configuration changes, where nodes are impacted FMs with their invalid configurations.

Our work brings the following main contributions. We present a novel approach to compute feature change impact on existing configurations of an MPL. We provide a prototype tool supporting our approach, available for download.[4] We demonstrate the applicability of the approach by applying it to a concrete case-study executed in cooperation with our industrial partner.

4.2 Approach

We describe here first how the model is built. Then, we show how we enrich the model with inferred information and finally the steps taken for simulating

[4] The tool is available at http://swerl.tudelft.nl/bin/view/NicolasDintzner/WebHome

the effects of a feature change on existing configurations. An overview of the different steps are shown in Figure 3.

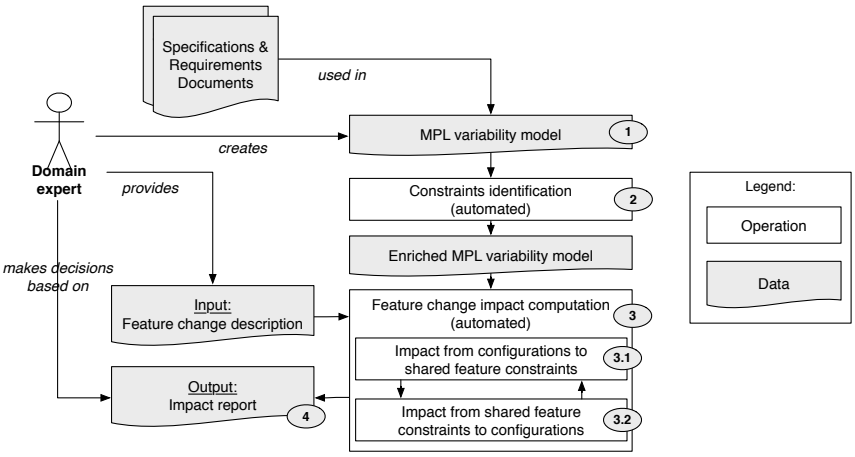

Fig. 3. Approach overview

Step 1: Describe subsystem variability. The first step of our approach consists in modelling the various subsystems using FM notation. This operation is done by domain experts, using existing documentation. When a subsystem uses a feature that has already been described in another subsystem, we reference it instead of creating a new one [1]. We associate with each FM its known configurations.

Step 2: Enrich the model with inferred composition rules. Once all FMs and configurations have been described, we use the configurations to infer how pairs of subsystems can be combined. We identify, in FMs sharing features, which features are shared and then create a list of existing partial configurations containing only them. Partial configurations appearing in existing configurations of both FMs constitute the whitelist of partial configurations enabling composition of configurations between the involved FMs. For two given FMs, the number of feature involved in shared feature constraints is equal to the number of features shared between them. Those partial configurations are the *shared feature constraints* relating pairs of FMs: two configurations, from two different FMs sharing features, are "compatible" if they contain exactly the same shared features. In order to apply such heuristic, shared feature constraints must be generated between every pairs of FMs sharing features. An example of such constraints is shown in Figure 4, where FMs 1 and 2 share features E and D.

Step 3: Compute the impact of a feature change. We use the enriched model to perform feature change impact computation at the request of domain experts. A feature change can be any modification of a FM (add/remove/move/modify features and constraints) or a change in available configurations (add/remove).

We assess the impact of the change of the configurations of a modified FM by re-validating them with respect to the updated FM, as suggested in [7]. This gives us a first set of invalid configurations that we use as a starting point for the propagation heuristic.

Step 3.1: Compute impact of configuration changes on shared feature constraints. We evaluate how a change of configuration of a FM affects the shared feature constraints attached to it. If a given shared feature constraint is not satisfied by at least one configuration of the FM then it is invalidated by the change. For each FM affected by a configuration change, we apply the reasoning presented in Algorithm 1. In the case a change does not modify existing configurations, this step will tell us that all existing constraints are still valid, but some can be added. Otherwise, if all configurations matching a constraint have been removed then that constraint is considered invalid (i.e. does not match a possible combination of configurations). Given a list of invalid shared feature constraints and the FMs to which it refers to, we can execute Step 3.2. If no shared feature constraints are modified, the computation stops here.

```
Data: a FM fm with an updated set of configurations
Result: a list of invalidated shared feature constraints lInvalidConstraints
foreach shared feature constraint of fm: sfc do
    allowedFeatures ← selected features of sfc;
    forbiddenFeatures ← negated features of sfc;
    foreach configuration of fm: c (list of feature names) do
        if allowedFeatures ⊂ c then
            if c ∩ forbiddenFeatures == ∅ then
                c is compliant;
            end
        end
    end
    if no compliant configuration found then
        add sfc to lInvalidConstraints;
    end
end
```

Algorithm 1: Configuration change propagation

Step 3.2: Compute impact of shared feature constraint changes on configurations. Given a set of invalid shared feature constraints obtained in the previous step, we evaluate how this invalidates other FMs configurations. If a configuration of an FM does not match any of the remaining shared feature constraints, it can no longer be combined with configurations of other FMs and is considered invalid. We apply the operations described in Algorithm 2. If any configuration is invalidated, we use the output of this step to re-apply Step 3.1.

Step 4: Consolidate results. We capture the result of the computation as a tree of changes. The first level of the tree is always a set of configuration

```
Data: fm: a FM with updated shared feature constraints
Result: lInvalidConfs: a list of invalidated configurations of fm
foreach configuration of fm: c (list of feature names) do
    foreach shared feature constraint of fm: sfc do
        allowedFeatures ← selected features of sfc;
        forbiddenFeatures ← negated features of sfc;
        if allowedFeatures ⊂ c then
            if c ∩ forbiddenFeatures == ∅ then
                c is compliant;
            end
        end
    end
    if no compliant constraint found then
        add c to lInvalidConfs;
    end
end
```

Algorithm 2: Shared feature constraint change propagation

changes. If more than one FM is touched by the initial change (e.g. removal of a shared feature) then we have a multi-root tree. Each configuration change object describes the addition or removal of any number of configurations. If a configuration change triggered a change in shared feature constraints, a shared feature constraint change is added as its child. A shared feature constraint change references the two FMs involved and any number of constraints that were added or removed. The configuration changes following this shared feature constraint modification are then added as a child "configuration change object". This structure allows us to describe the path taken by the impact propagation through the different FMs.

4.3 Example

Let us consider the example shown in Figure 4, where two FMs share two features: D and E. The model is enriched with the "shared feature constraints" deduced from existing configurations. Those constraints state that, for a configuration of FM1 and FM2 to be combined, both of them need to have shared features that are either (E,D), (D, not E) and (not E, not D). The resulting data structure is shown on the left hand side of Figure 4.

We consider the following change: Configuration 1.2 is removed, operation marked as 1 in Figure 4. We apply the algorithm described in Step 3.1, using FM1 as a input, and with Configurations 1.1 and 1.3 (all of them except the removed one) and the associated 3 shared feature constraints. For Constraint 1, the allowed features are "E" and "D", and there are no forbidden features. We search for existing configurations of FM1 containing both "E" and "D" among Configurations 1.1 and 1.3. We find that Configuration 1.3 satisfies this constraint. The Constraint 2 (allowing "D" and forbidding "E") is not matched

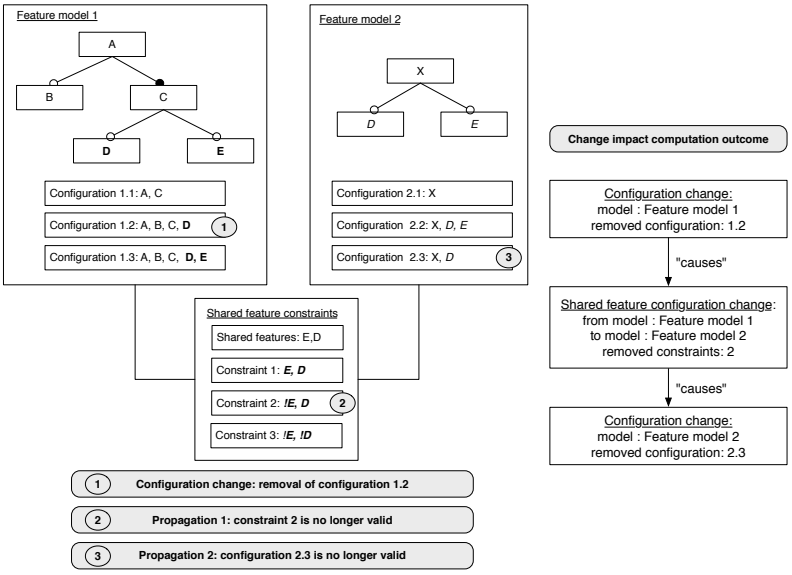

Fig. 4. Change impact propagation example

by any configurations, since the only configuration containing "D" and not "E" is Configuration 1.2 has been removed. Constraint 3 forbidding features "D" and "E" is satisfied by Configuration 1.1. The resulting list of invalid constraints contains only one element: Constraint 2 (marked as operation 2 in the diagram).

We then apply 2 presented in Step 3.2 to assess the effect of that change on the configurations of other FMs (FM2 only in this case). With the remaining Constraints 1 and 3, we run through the configurations of FM2 to identify which configurations no longer satisfy any constraints. We find that Configuration 2.1 satisfies Constraint 3 (does not contain "D" nor "E"), and Configuration 2.2 satisfies Constraint 1 (contains both "E" and "D"). However, configuration 2.3 does not satisfy any of the remaining constraints and for this reason, is marked as invalid (shown as operation 3 on the diagram).

On the right hand side of Figure 4, we present the resulting tree (a branch in this case). The intial change (removal of configuration 1.2 of FM1) is captured by the first "configuration change" object. Changes to shared features constraints are directly attached to this configuration change: the "shared feature configuration change" object. Finally, the last node of the tree is the invalidation of Configuration 2.3 of FM2.

4.4 Scalability aspects

The initial step of our approach replicates what Heider suggests in [7]: reinstantiating existing configurations. Such approaches are known as *product-based* approaches [20]. They have known drawbacks: as the number of configurations

and features increases, the solution does not scale. By placing ourselves in an MPL enviroment, we have small to medium size FMs to analyze and perform this type of operation only on individual FMs.

Our composition heuristic focuses on composition of configurations (as opposed to composition of FMs). Once the local product-based approach is used, we rely on it to identify broken compositions of configurations across the FMs without having to revalidate any configurations against the FMs. This last step can be viewed as a *family-based* analysis of our product line [20], where we validate a property over all members of a PL. We store information relative to shared feature constraints on the model itself. With this information, applying the heuristic to an MPL amounts to searching specific character strings in an array, which is much faster than merging models or validating complete configurations.

4.5 Prototype implementation

We implemented a prototype allowing us to import FMs into a database, enrich the model and run feature change impact computations. The choice of using a database was motivated by potential integration with other datasources. Since FMs are mostly hierarchical structures, we use Neo4j.[5] Our Neo4j schema describes the concepts of feature model, feature, configuration and shared feature constraint with their relationships as described in the previous section. This representation is very similar to other FM representations such as [21] with one exception. The mandatory, optional or alternative nature of a feature is determined by its relationship with its parent; as opposed to be a characteristic of the feature itself. This allows to have an optional feature in a FM, referenced by another FM as part of an alternative.

We leverage the Neo4j *Cypher* query language to retrieve relevant data: shared features, configurations containing certain features as well as interconnected feature models and the features which links them. We use FeatureIDE [21] as a feature model editor tool. We import models in their xml format into our database using a custom java application. A basic user interface allows us to give the name of a feature to remove, run the simulation, and view the result.

5 Industrial case study

As mentioned in Section 3, this paper is motivated by an industrial case study proposed by our partner. The end-goal of this case study is to assess the applicability of our approach in an industrial context. To do so, we reproduce a past situation where a change modified the behaviour of some products of their product line on which a 3rd party product was relying, and where the impact was detected late in the development process. We present and discuss the main steps of our approach and their limitations when applied in an industrial context:

[5] http://www.neo4j.org

the construction of the model, the feature change impact computation with its result, and the performance of our prototype implementation.

5.1 Modelling a X-ray MPL

We start by gathering specification documents of the main subsystems identified in Section 3, as well as 3rd party product compatibility specifications. With the domain experts, we identify relevant components and existing configurations of each subsystem. Using this information, we model the three interfaces and three subsystems presented in Figure 2 as six distinct feature models (FMs). The three interfaces are (i) the video/data transfer interface (data and video transfer capabilities), (ii) the video export interface specifying possible resolutions and refresh rates, and finally (iii) the display interface representing a choice in monitor and display modes. 3rd party product interface usages are modeled as the configurations associated to those FMs. The three subsystems of the x-ray machine are (i) the data/video transfer subsystem, (ii) the video chain used to transport images from a source to a display, and finally (iii) display subsystem. Configurations of those subsystems are the concrete products available to customers (between 4 and 11 configurations per FM). Each FM contains between 10 and 25 features, with at most 5 cross-tree constraints. The "data transfer", "video chain", and "screen" FMs share features relating to hardware components, and reuse features from interface FMs. We use FeatureIDE to create FMs and configurations. We then import them into a Neo4J database and use our prototype implementation to generate the necessary shared feature constraints as described in Section 4.

The main challenge of this phase is to ensure that shared features represent the same concept in all FMs. For instance, a feature "cable" refers to one specific cable, in a specific context, and must be understood as such in all FMs including it. Misreferencing features will lead to incorrect shared feature constraints and incorrect change impact analysis results. We mitigated this effect by carefully reviewing FMs and shared features with domain expert.

5.2 Simulating the change

We studied the effect of the removal of a hardware component used to import video into the system. To simulate this with our prototype, we provide our tool with the name of the feature to remove ("Connection box 1"). "Connection box 1" is included in both the "data/video transfer" and "video chain" FMs, so its removal directly impacts those two FMs. The tool re-instantiates all configurations of those two FMs and find that 6 configurations of the "video chain" FM, and 1 from the "data transfer" FM are invalid. Then, the prototype executes the propagation heuristic. A shared feature constraint between the "data transfer" and "data transfer interface" FMs is no longer satisfied by any configuration of the "data transfer" FM, and is now invalid. Without this shared feature constraint, one configuration of the "data transfer interface" FM can no longer be combined with the "data transfer" FM and is considered as invalid. The removal of a configuration in an interface FM tells us that the compatibility

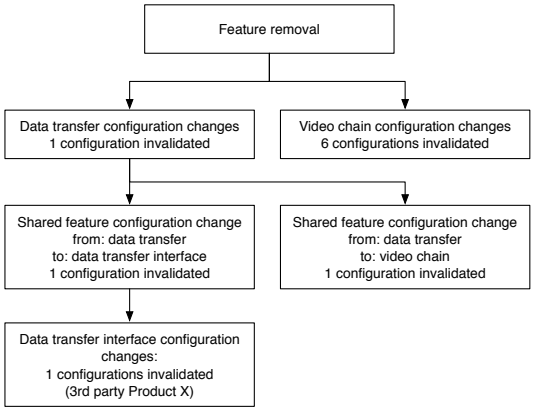

Fig. 5. Output of the feature removal simulation

with one 3rd party product is no longer possible. The modifications of the "data transfer" FM also invalidated a shared feature constraint existing between the "data transfer" and "video chain" FMs. However, the change of "shared feature constraint" did not propagate further; the configurations that it should have impacted had already been invalidated by a previous change.

The result of this impact analysis, reviewed with 3 domain experts, showed the impact of interfaces that had been problematic in the past. We ran several other simulations on this model (removal of features, removal of configurations). On each occasion, the result matched the expectations of domain experts - given the data included in the model. In this context, the approach proved to be both simple and successful. This being said, by using information from existing configurations, we over-constrain inter-FMs relationships. If a shared optional feature is present in all configurations of a given FM, it will be seen as mandatory during impact computation. However, if a feature is present in all of existing configurations, it is mandatory with respect to what is available - as opposed to mandatory in the variability model. As long as we reason about existing configurations only, using inferred shared feature constraints should not influence negatively the result of the simulation.

5.3 Performance analysis

We provide here a qualitative overview of performance measurements that were performed during this case study. For our main scenario, our approach checked all configurations of 2 of the FMs, and the change propagated to 2 others. 2 of the 6 FMs did not have to be analyzed. In this specific context, our implementation provided results in less than a few seconds, regardless of the scenario that was ran. We then artificially increased the size of the models (number of features and number of configurations) to evaluate how it influences the computation time of the propagation algorithm. Given a set of invalid configurations, we measure

how long it takes to assess the impact on one connected FM. For 2 FMs with 20 features each and 20 configurations each, sharing 2 features, the propagation from 1 FM to the other and impact its configurations takes approximately 450ms. With 200 configurations in each FMs, the same operation takes 1.5s; and up to 2.5s for 300 configurations.

During the industrial case study, the performance of the prototype tool was sufficient to provide almost real-time feedback to domain engineers. The size of the models and the number of configurations affect negatively the computation time of the change impact analysis, because the first step of our approach is product-based: we do check all configurations of the initially impacted FMs. However, using an MPL approach, individual FMs are meant, by design, to be relatively small. Then, computing the propagation of those changes, if any, depends on the number of affected FMs as defined by our propagation heuristic. The heuristic itself is the validation of a property over all members of the product family ("family-based" approach), so its performance is less influenced by model size [20]. This operation consists in searching for strings in an array, which should remain manageable even for large models. Our naive implementation, using Neo4j, already provided satisfactory performance.

5.4 Threats to validity

With respect to *internal validity*, the main threat relates to the construction of the models used for the industrial case study. We built the different FMs and configurations of the case study using existing documentation while devising the approach. To avoid any bias in the model construction, we reviewed the models several times with domain experts, ensuring their representativeness.

Threats to *external validity* concern the generalisation of our findings. For this study, we used only the most basic FM notation [10]. Our approach should be applicable using more complex notations as long as those notation do not change the representation of the configurations (list of feature names, where each name appear once). If, for instance, we use a cardinality-based notation, the heuristic will have to be adapted to take this cardinality into account. The extracted information from existing configurations was sufficient for this case study, but more complex relationships between FMs might not have been encountered. Applying our approach on a different PL would confirm or infirm this.

6 Related work

The representation of variability in very large systems, using multiple FMs, has been studied extensively during the past few years. Several composition techniques have been devised. Composition rules can be defined at an FM level, specifying how the models should be recombined for analysis. Otherwise, cross-FM constraints can be defined. Examples can be found in the work of Schirmeier [16] and Acher [1, 2]. In our context, we chose not to follow those approaches as we do not know a priori the over-arching relationships between FMs, nor can we

define cross-FM constraints since we work with partial information. Moreover, those techniques would then require us to re-compose models before validating the various configurations which, as noted in [6], is complex to automate. Recent work on MPLs showed that there is a need to specialise feature models to segregate concerns in MPL variability models. Reiser et al. [15] propose the concept of "context variability model" for multi-product lines, which describes the variability of the environment in which the end product resides. In our study, we classified our FMs as either interface or subsystem. This classification also allows us to qualify the configurations (as interface usage or product implementation), which proved to be sufficient for our application. Schröter et al. present the idea of interface FMs where specific FMs involved in an MPL act as interfaces between other FMs [18]. They propose a classification of the characteristics that can be captured by such models (syntactic, behavioral, and non-functional). While we did not use this approach directly, we noted that for non-interface FMs, we used specific branches of the model to organize reused shared features. It is interesting to note that the designs of (non-interface) FMs share a common structure. We used specific branches of their respective FM to organise features shared with interface FMs. Doing so, we specialized a branch of a FM instead of creating dedicated FMs and we do not restrict the type of features it contains (functional and non-functional alike).

Heider et al. proposed to assess the effect of a change on a variability model by re-instantiating previously configured products [7], and thus validating non-regression. Our approach applies similar principles, as we will consider a change safe as long as the existing products can be re-derived. We apply those concepts in a multi-product line environment, where change propagation is paramount. Thüm et al. [19] proposed to classify changes occurring on feature models based on their effect on existing product configurations. The change is considered as a "generalisation" if the set of valid products has been extended, "specialisation" if it has been reduced, "refactoring" if it has not changed, and "arbitrary edit" in all other cases (when some configurations were removed and others added). This initial classification gave us some insight into the potential impact of a change, but only for a single FM. Their methodology could be applied during the initial step of our approach to identify changes that do not affect existing configurations, avoiding extra computation later on.

7 Conclusion

Understanding the full extent of the impact of a change on a complex and highly variable product is a difficult task. The main goal of this research is to facilitate the evolution of such systems by assisting domain experts in assessing the effects of changes on multi-product line variability models. In this paper, we presented an approach to compute the impact of a feature change on a multi-product line for non-regression purposes, leveraging information contained in existing product configurations to infer feature model composition constraints. We described how our modelling approach can be used in a practical context,

using an industrial case and provide a qualitative review of the performance of our prototype tool. With partial information, we were able to accurately identify which configurations of an MPL were rendered invalid by a feature change.

As industrial products grow more complex and become more variable, managing their evolution becomes increasingly difficult. Approaches supporting domain experts' activities will have to be adapted to meet new challenges. As a step in that direction, we released our implementation as an open source project [6] as well as the dataset we used for the performance evaluation. We then plan to integrate it into existing feature modelling tools. We intend to explore how we can make the best use of the promising graph database technologies such as Neo4J for feature model checking. With such technology, we will be in a position to consider more complex models, with potentially more complex FM composition constraints, further facilitating the design, analysis and maintenance of highly variable systems.

8 Acknowledgements

This publication was supported by the Dutch national program COMMIT and carried out as part of the Allegio project under the responsibility of the Embedded Systems Innovation group of TNO in partnership with Philips Healthcare.

References

1. Acher, M., Collet, P., Lahire, P., France, R.: Comparing approaches to implement feature model composition. In: Kühne, T., Selic, B., Gervais, M.P., Terrier, F. (eds.) Proc. of the 6th European Conf. on Modelling Foundations and Applications. pp. 3–19. ECMFA '10, Springer Berlin Heidelberg (Jun 2010)
2. Acher, M., Collet, P., Lahire, P., France, R.: Managing multiple software product lines using merging techniques. Research report ISRN I3S/RR–20 10 - 06 –FR, Laboratoire d'Informatique de Signaux et Systèmes de Sophia Antipolis - UNSA-CNRS (2010)
3. Acher, M., Collet, P., Lahire, P., France, R.: Managing variability in workflow with feature model composition operators. In: Baudry, B., Wohlstadter, E. (eds.) Proc. of the 9th International Conf. on Software Composition. pp. 17–33. SC '10, Springer Berlin Heidelberg (Jul 2010)
4. Bagheri, E., Gasevic, D.: Assessing the maintainability of software product line feature models using structural metrics. Software Qual J 19(3), 579–612 (Sep 2011)
5. Czarnecki, K., Helsen, S., Eisenecker, U.: Staged configuration through specialization and multilevel configuration of feature models. Software Process: Improvement and Practice 10(2), 143–169 (Apr 2005)
6. Hartmann, H., Trew, T.: Using feature diagrams with context variability to model multiple product lines for software supply chains. In: Proc. of the 12th International Software Product Line Conference. pp. 12–21. SPLC '08, IEEE (Sep 2008)

[6] The tool is available at http://swerl.tudelft.nl/bin/view/NicolasDintzner/WebHome

7. Heider, W., Rabiser, R., Grünbacher, P., Lettner, D.: Using regression testing to analyze the impact of changes to variability models on products. In: Proc. of the 16th International Software Product Line Conference. p. 196–205. SPLC '12, ACM (2012)
8. Heider, W., Vierhauser, M., Lettner, D., Grunbacher, P.: A case study on the evolution of a component-based product line. In: 2012 Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) and European Conference on Software Architecture (ECSA). pp. 1 –10 (Aug 2012)
9. Holl, G., Thaller, D., Grünbacher, P., Elsner, C.: Managing emerging configuration dependencies in multi product lines. In: Proc. of the 6th International Workshop on Variability Modeling of Software-Intensive Systems. p. 3–10. VaMoS '12, ACM (2012)
10. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (FODA) feasibility study. Tech. rep., Software Engineering Institute, Carnegie Mellon University (1990)
11. Krueger, C.: New methods in software product line development. In: Proc. of the 10th International Software Product Line Conference. pp. 95–99. SPLC '06, IEEE Computer Society (Aug 2006)
12. Ommering, R.C.v., Bosch, J.: Widening the scope of software product lines - from variation to composition. In: Proc. of the 2nd International Conference on Software Product Lines. p. 328–347. SPLC '02, Springer-Verlag (2002)
13. Paskevicius, P., Damasevicius, R., Štuikys, V.: Change impact analysis of feature models. In: Skersys, T., Butleris, R., Butkiene, R. (eds.) Information and Software Technologies, pp. 108–122. No. 319 in Communications in Computer and Information Science, Springer Berlin Heidelberg (2012)
14. Pohl, K., Böckle, G., Linden, F.J.v.d.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag (2005)
15. Reiser, M.O., Weber, M.: Multi-level feature trees: A pragmatic approach to managing highly complex product families. Req. Eng. 12(2), 57–75 (May 2007)
16. Schirmeier, H., Spinczyk, O.: Challenges in software product line composition. In: Proc. of the 42nd Hawaii International Conference on System Sciences. HICSS '09, IEEE (Jan 2009)
17. Schmid, K.: Variability support for variability-rich software ecosystems. pp. 5–8. IEEE (May 2013)
18. Schröter, R., Siegmund, N., Thüm, T.: Towards modular analysis of multi product lines. In: Proc. of the 17th International Software Product Line Conference Colocated Workshops. p. 96–99. ACM (2013)
19. Thuem, T., Batory, D., Kaestner, C.: Reasoning about edits to feature models. In: Proc. of the 31st International Conference on Software Engineering. p. 254–264. ICSE '09, IEEE Computer Society (2009)
20. Thüm, T., Apel, S., Kästner, C., Schaefer, I., Saake, G.: A classification and survey of analysis strategies for software product lines. ACM Comput. Surv. 47(1), 6:1–6:45 (Jun 2014)
21. Thüm, T., Kaestner, C., Benduhn, F., Meinicke, J., Saake, G., Leich, T.: FeatureIDE: An extensible framework for feature-oriented software development. Science of Computer Programming 79(0), 70–85 (2014)

TUD-SERG-2014-018
ISSN 1872-5392